21世纪高等学校规划教材｜计算机科学与技术

数据库原理

严冬梅 编著

清华大学出版社

VISIT…

LANZAROTE

Caliente.com

21世纪高等学校规划教材·计算机科学与技术

数据库原理

严冬梅 主 编

清华大学出版社
北 京

内 容 简 介

本书以关系数据库系统为核心，全面介绍了数据库系统的基本原理。全书共 10 章，主要内容包括数据库系统基本概念、关系数据模式、关系数据库标准语言 SQL、关系数据库理论、查询优化、数据库保护、数据库应用系统设计、数据库编程、数据库产品及数据库技术新发展。本书中所涉及的例子均以学生学习过程为主线，每章后均附有习题，习题答案可从网站下载。为了配合教学和学生自学，本书配有制作精美的 PPT 课件。同时，本书还有配套教材《数据库应用与实践指导》对实验环节进行指导。

本书可作为普通高等院校计算机及相关学科的数据库课程教材，也可作为数据库技术的自学教材和参考书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

数据库原理/严冬梅编著. —北京：清华大学出版社，2011.9
（21 世纪高等学校规划教材·计算机科学与技术）
ISBN 978-7-302-26174-2

Ⅰ. ①数…　Ⅱ. ①严…　Ⅲ. ①数据库系统　Ⅳ. ①TP311.13

中国版本图书馆 CIP 数据核字(2011)第 136855 号

责任编辑：刘向威　李玮琪
责任校对：焦丽丽
责任印制：

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185×260　　**印　　张**：14.5　　**字　　数**：352 千字
版　　次：2011 年 9 月第 1 版　　**印　　次**：2011 年 9 月第 1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：041388-01

编审委员会成员

（按地区排序）

清华大学	周立柱	教授
	覃　征	教授
	王建民	教授
	冯建华	教授
	刘　强	副教授
北京大学	杨冬青	教授
	陈　钟	教授
	陈立军	副教授
北京航空航天大学	马殿富	教授
	吴超英	副教授
	姚淑珍	教授
中国人民大学	王　珊	教授
	孟小峰	教授
	陈　红	教授
北京师范大学	周明全	教授
北京交通大学	阮秋琦	教授
	赵　宏	教授
北京信息工程学院	孟庆昌	教授
北京科技大学	杨炳儒	教授
石油大学	陈　明	教授
天津大学	艾德才	教授
复旦大学	吴立德	教授
	吴百锋	教授
	杨卫东	副教授
同济大学	苗夺谦	教授
	徐　安	教授
华东理工大学	邵志清	教授
华东师范大学	杨宗源	教授
	应吉康	教授
东华大学	乐嘉锦	教授
	孙　莉	副教授

浙江大学	吴朝晖	教授
	李善平	教授
扬州大学	李　云	教授
南京大学	骆　斌	教授
	黄　强	副教授
南京航空航天大学	黄志球	教授
	秦小麟	教授
南京理工大学	张功萱	教授
南京邮电学院	朱秀昌	教授
苏州大学	王宜怀	教授
	陈建明	副教授
江苏大学	鲍可进	教授
中国矿业大学	张　艳	教授
武汉大学	何炎祥	教授
华中科技大学	刘乐善	教授
中南财经政法大学	刘腾红	教授
华中师范大学	叶俊民	教授
	郑世珏	教授
	陈　利	教授
江汉大学	颜　彬	教授
国防科技大学	赵克佳	教授
	邹北骥	教授
中南大学	刘卫国	教授
湖南大学	林亚平	教授
西安交通大学	沈钧毅	教授
	齐　勇	教授
长安大学	巨永锋	教授
哈尔滨工业大学	郭茂祖	教授
吉林大学	徐一平	教授
	毕　强	教授
山东大学	孟祥旭	教授
	郝兴伟	教授
中山大学	潘小轰	教授
厦门大学	冯少荣	教授
仰恩大学	张思民	教授
云南大学	刘惟一	教授
电子科技大学	刘乃琦	教授
	罗　蕾	教授
成都理工大学	蔡　淮	教授
	于　春	副教授
西南交通大学	曾华桑	教授

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施“高等学校本科教学质量与教学改革工程”（简称“质量工程”），通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程（群）加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合“质量工程”的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的“高等学校教学质量与教学改革工程精品课程建设工作”，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了“清华大学出版社教材编审委员会”（以下简称“编委会”），旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。“编委会”成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，“编委会”一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上。精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性（新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量）、先进性（对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向）、示范性（教材所体现的课程体系具有较广泛的辐射性和示范性）和一定的前瞻性。教材由个人申报或各校推荐（通过所在高校的“编委会”成员推荐），经“编委会”认真评审，最后由清华大学出版

社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括：

（1）21世纪高等学校规划教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算机应用类教材。

（2）21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

（3）21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

（4）21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

（5）21世纪高等学校规划教材·信息管理与信息系统。

（6）21世纪高等学校规划教材·财经管理与应用。

（7）21世纪高等学校规划教材·电子商务。

（8）21世纪高等学校规划教材·物联网。

清华大学出版社经过二十多年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人：魏江江
E-mail：weijj@tup.tsinghua.edu.cn

前　言

1. 编写目的

数据库技术是数据管理的技术，是计算机科学的重要组成部分。数据库技术自20世纪60年代产生至今已有近50年的历史，经历了从以网状模型和层次模型为特征的第一代数据库系统、以关系模型为基础的第二代数据库系统，到今天的以面向对象为主要特征、融合多种新技术的第三代数据库系统的演变过程，产生了一个新的、巨大的软件产业，推动了计算机技术的应用和发展。目前，不仅在各种机型上配有数据库管理系统，各行各业的信息系统，甚至是Internet上的信息系统都离不开数据库的支持。因此，数据库已成为信息社会的重要基础设施。数据库的应用领域非常广，已经拓展到商业、医疗保健、教育、政府组织、图书馆、军事、工业控制等诸多领域。因此，数据库技术已成为计算机技术、控制技术、信息技术等相关专业的必修课程，也是从事相关领域的工程技术人员所必须具备的专业知识。

本书由浅入深、循序渐进、理论与实践并重，力求让读者通过本书的学习，能对数据库技术有一个比较全面的了解，掌握数据库理论的基本知识，对数据库应用系统的开发有初步的了解。

2. 内容介绍

本书共分为十章，各章的主要内容如下：

第1章概要介绍数据库系统的基本概念和基本结构，包括数据管理的发展、数据库管理系统及其相关概念、数据库系统结构、数据模型和关系数据库的基本概念。

第2章系统地讲解关系数据库的重要概念，对关系模型进行描述；并通过实例详细讲解关系代数，包括传统的并、交、差、笛卡儿积集合操作和特殊的选择、投影、连接、除运算。

第3章详细介绍关系数据库标准语言SQL的组成、功能和特点，重点介绍数据定义、数据更新、数据查询、数据控制语言。

第4章全面介绍关系数据库，包括关系模型、关系模式、关系代数以及关系数据库规范化理论。

第5章介绍关系型数据库系统的查询优化技术，主要包括RDBMS查询的基本处理过程、查询优化的基本概念和基本方法，并给出一些在实际应用中的查询优化方法。

第6章详细介绍数据库保护，包括数据库安全性、完整性、并发控制和数据库恢复。

第7章介绍关系数据库设计，着重介绍数据库设计步骤，特别是数据库概念结构设计和逻辑结构设计。

第8章介绍嵌入式SQL语句、存储过程以及如何连接、访问数据库。

第9章简要介绍市场上广泛使用的数据库产品，包括SQL Server、ORACLE、MySQL、

Sybase 和 DB2。

第 10 章介绍数据库技术的最新发展，包括面向对象数据库、分布式数据库、Web 数据库、数据仓库和电子商务数据库。

3. 适用对象

本书可作为高等学校本科和研究生的数据库技术相关课程的教材或参考书，也可供广大从事数据库技术研究、开发与应用的工程技术人员参考。

4. 建议学时安排

建议课时安排

序 号	内 容	学 时
1	第 1 章 数据库概述	6
2	第 2 章 关系数据库系统	4
3	第 3 章 关系数据库标准语言 SQL	6
4	第 4 章 关系数据理论	4
5	第 5 章 查询优化	2
6	第 6 章 数据库保护	8
7	第 7 章 数据库设计	6
8	第 8 章 数据库编程	4
9	第 9 章 数据库产品简介	2
10	第 10 章 数据库技术新发展	4
11	复习	2
	总计	48

5. 编写情况

参与本书编写的所有人员均为天津财经大学的教师及研究生。本书由严冬梅编著，负责统编、修改及总撰定稿。其中第 1、2、4 章由严冬梅编写，第 3 章由饶俊、严冬梅编写，第 5 章由但志广编写，第 6、7 章由陈立君编写，第 8 章由宋丽红编写，第 9、10 章由张铠、李玉芝编写。

在本书的编写过程中，得到了鲁城华的大力协助，还得到了清华大学出版社的鼎力支持和帮助，在此表示衷心的感谢！本书的编写还参考了很多国内外相关的资料，对所有原作者也表示诚挚的谢意。

由于水平所限，书中难免存在不足和错误，敬请广大读者提出建议和批评意见。

目 录

第 1 章 数据库概述 …… 1

1.1 计算机数据管理的发展 …… 1
1.1.1 数据管理 …… 1
1.1.2 数据库技术的产生和发展 …… 2
1.2 数据库管理系统 …… 5
1.2.1 数据库管理系统的定义 …… 5
1.2.2 数据库管理系统的功能 …… 5
1.3 数据库系统 …… 6
1.3.1 数据库系统的定义 …… 6
1.3.2 数据库系统的组成 …… 6
1.3.3 数据库系统的模式 …… 7
1.3.4 数据库语言 …… 10
1.4 数据模型 …… 10
1.4.1 数据处理的三个领域 …… 11
1.4.2 数据模型的要素 …… 14
1.4.3 数据模型的分类 …… 14
1.5 本章小结 …… 18
1.6 习题 …… 19
1.6.1 名词解释 …… 19
1.6.2 简答题 …… 19
1.6.3 用 E-R 图表示概念模型 …… 19

第 2 章 关系数据库系统 …… 21

2.1 关系数据结构 …… 21
2.1.1 关系及相关概念 …… 21
2.1.2 关系模式 …… 24
2.1.3 关系数据库 …… 25
2.2 关系操作集合 …… 26
2.2.1 基本关系操作 …… 26
2.2.2 关系数据语言分类 …… 26
2.3 完整性约束 …… 27
2.3.1 实体完整性 …… 27

2.3.2 参照完整性 …… 28
2.3.3 用户定义完整性 …… 29
2.4 关系代数 …… 30
2.4.1 传统的集合运算 …… 30
2.4.2 专门的关系运算 …… 32
2.4.3 综合算例 …… 36
2.5 本章小结 …… 39
2.6 习题 …… 39
2.6.1 名词解释 …… 39
2.6.2 简答题 …… 39
2.6.3 综合题 …… 40

第3章 关系数据库标准语言SQL …… 41

3.1 SQL简介 …… 41
3.1.1 SQL的特点 …… 41
3.1.2 SQL语言简介 …… 42
3.2 SQL数据定义功能 …… 44
3.2.1 创建、删除模式 …… 44
3.2.2 创建、删除、修改基本表 …… 45
3.2.3 创建、删除、修改索引 …… 47
3.3 SQL数据查询功能 …… 48
3.3.1 单表查询 …… 48
3.3.2 连接查询 …… 53
3.3.3 嵌套查询 …… 55
3.3.4 集合查询 …… 57
3.4 SQL数据操纵功能 …… 59
3.4.1 插入数据 …… 59
3.4.2 修改数据 …… 60
3.4.3 删除数据 …… 60
3.5 视图 …… 62
3.5.1 定义和删除视图 …… 62
3.5.2 查询视图 …… 63
3.5.3 更新视图 …… 64
3.6 数据控制 …… 65
3.6.1 授权 …… 66
3.6.2 收回权限 …… 66
3.7 本章小结 …… 67
3.8 习题 …… 67
3.8.1 名词解释 …… 67

3.8.2 简答题 …… 67
3.8.3 综合题 …… 67

第 4 章 关系数据理论 …… 69

4.1 数据存储异常 …… 69
4.1.1 关系模式设计概述 …… 69
4.1.2 关系模式的数学表示 …… 70
4.1.3 实例分析 …… 70
4.2 函数依赖 …… 73
4.2.1 函数依赖的一般概念 …… 73
4.2.2 Armstrong 公理系统 …… 76
4.3 关系模式的规范化 …… 77
4.3.1 第一范式 …… 77
4.3.2 第二范式 …… 78
4.3.3 第三范式 …… 79
4.3.4 BC 范式 …… 79
4.3.5 多值依赖和第四范式 …… 80
4.3.6 多值依赖和第五范式 …… 82
4.3.7 规范化过程小结 …… 82
4.4 关系模式的分解 …… 83
4.4.1 关系模式分解的标准 …… 83
4.4.2 无损连接性 …… 84
4.4.3 保持函数依赖 …… 84
4.5 在实际数据库设计中关系规范化的应用 …… 84
4.5.1 关系规范化的基本原则 …… 84
4.5.2 关系规范化的实际应用 …… 85
4.6 本章小结 …… 87
4.7 习题 …… 87
4.7.1 名词解释 …… 87
4.7.2 简答题 …… 87
4.7.3 综合题 …… 87

第 5 章 关系查询处理与优化 …… 89

5.1 查询优化概述 …… 89
5.1.1 查询中遇到的问题 …… 89
5.1.2 查询优化的必要性 …… 90
5.1.3 查询优化的可行性 …… 91
5.2 查询处理过程 …… 92
5.2.1 查询分析 …… 92

5.2.2 查询检查 …… 92
5.2.3 查询优化 …… 93
5.2.4 查询执行 …… 93
5.3 查询优化方法 …… 93
5.3.1 代数优化 …… 93
5.3.2 物理优化 …… 96
5.4 实际应用中的查询优化 …… 99
5.4.1 基于索引的优化 …… 99
5.4.2 查询语句的优化 …… 100
5.5 本章小结 …… 102
5.6 习题 …… 103
5.6.1 简答题 …… 103
5.6.2 综合题 …… 103

第 6 章 数据库保护 …… 104

6.1 数据库安全性 …… 104
6.1.1 数据库安全性概述 …… 104
6.1.2 数据库安全性策略 …… 104
6.2 数据库完整性 …… 110
6.2.1 完整性概述 …… 110
6.2.2 完整性约束条件 …… 111
6.2.3 完整性控制 …… 112
6.3 数据库并发控制 …… 114
6.3.1 事务概述 …… 114
6.3.2 并发控制概述 …… 115
6.3.3 封锁 …… 116
6.3.4 活锁与死锁 …… 117
6.4 数据库恢复 …… 119
6.4.1 数据库恢复概述 …… 119
6.4.2 故障的种类 …… 119
6.4.3 故障恢复 …… 120
6.4.4 恢复策略 …… 123
6.5 本章小结 …… 124
6.6 习题 …… 124
6.6.1 名词解释 …… 124
6.6.2 简答题 …… 125
6.6.3 综合题 …… 125

第 7 章 数据库设计 …… 126
7.1 数据库设计概述 …… 126
7.1.1 数据库设计的内容 …… 126
7.1.2 数据库设计的方法 …… 127
7.1.3 数据库设计的步骤 …… 127
7.2 需求分析 …… 128
7.2.1 需求分析的任务 …… 128
7.2.2 需求分析的步骤和方法 …… 129
7.2.3 需求分析注意的问题 …… 130
7.3 概念结构设计 …… 130
7.3.1 概念结构设计的方法与步骤 …… 130
7.3.2 数据抽象 …… 131
7.3.3 采用 E-R 方法的数据库概念结构设计 …… 132
7.4 逻辑结构设计 …… 135
7.4.1 E-R 图转换为数据模型 …… 135
7.4.2 关系规范化 …… 137
7.4.3 数据模型的优化 …… 137
7.4.4 设计外模式 …… 137
7.5 数据库物理设计 …… 138
7.5.1 数据库物理设计的内容和方法 …… 138
7.5.2 关系模式存取方法的选择 …… 139
7.5.3 确定数据库的存储结构 …… 140
7.6 数据库实施 …… 141
7.6.1 定义数据结构 …… 142
7.6.2 数据装载 …… 142
7.6.3 编制与调试应用程序 …… 143
7.6.4 数据库试运行 …… 143
7.6.5 数据库其他设计 …… 143
7.7 数据库运行和维护 …… 144
7.7.1 数据库的转储与恢复 …… 144
7.7.2 数据库的安全性与完整性维护 …… 145
7.7.3 数据库性能的监督与改进 …… 145
7.7.4 数据库的功能完善 …… 145
7.8 本章小结 …… 145
7.9 习题 …… 146
7.9.1 名词解释 …… 146
7.9.2 简答题 …… 146
7.9.3 综合题 …… 146

第 8 章 数据库编程 …… 147

8.1 嵌入式 SQL …… 147
8.1.1 嵌入式 SQL 的特点 …… 147
8.1.2 SQL 语言和宿主语言编程 …… 147
8.1.3 静态 SQL 编程 …… 150
8.1.4 动态 SQL 编程 …… 155
8.2 存储过程 …… 158
8.2.1 存储过程概述 …… 158
8.2.2 创建和执行存储过程 …… 159
8.2.3 管理存储过程 …… 163
8.2.4 系统存储过程 …… 164
8.3 本章小结 …… 165

第 9 章 数据库产品简介 …… 166

9.1 SQL Server …… 166
9.1.1 SQL Server 的简介 …… 166
9.1.2 SQL Server 的特点 …… 167
9.1.3 SQL Server 2008 的新特性 …… 168
9.1.4 应用程序访问 SQL Server 的实例 …… 171
9.2 Oracle …… 172
9.2.1 Oracle 的发展历程 …… 172
9.2.2 Oracle 的特点 …… 173
9.2.3 Oracle 的开发工具 …… 174
9.2.4 应用程序访问 Oracle 的实例 …… 175
9.3 MySQL …… 175
9.3.1 MySQL 简介 …… 176
9.3.2 MySQL 的特点 …… 176
9.3.3 MySQL 的开发工具 …… 178
9.4 Sybase …… 179
9.4.1 Sybase 数据库的发展史 …… 179
9.4.2 Sybase 数据库的特点 …… 179
9.4.3 Sybase 数据库的组成 …… 181
9.4.4 Sybase 数据库的开发工具 …… 181
9.5 DB2 …… 182
9.5.1 DB2 的发展历程 …… 182
9.5.2 DB2 的特点 …… 183
9.5.3 DB2 的开发工具 …… 183
9.5.4 应用程序访问 DB2 的实例 …… 184

9.6 本章小结 …… 184

第 10 章 数据库技术的新发展 …… 186

10.1 面向对象的数据库系统 …… 186
10.1.1 面向对象数据库系统的基本特征 …… 187
10.1.2 面向对象数据模型 …… 188
10.1.3 面向对象数据库语言 …… 189
10.1.4 对象关系数据库 …… 191
10.2 分布式数据库系统 …… 193
10.2.1 数据库系统体系结构 …… 193
10.2.2 分布式数据库系统的概念和特点 …… 194
10.2.3 分布式数据库的体系结构 …… 197
10.2.4 分布式数据库系统的分类 …… 197
10.3 Web 与数据库 …… 198
10.3.1 Web 数据库 …… 198
10.3.2 Web 数据库与传统数据库比较 …… 199
10.3.3 Web 服务器脚本程序与服务器的接口 …… 200
10.3.4 应用开发平台 …… 201
10.4 数据仓库 …… 202
10.4.1 数据仓库概述 …… 202
10.4.2 数据仓库的基本特性 …… 203
10.4.3 数据仓库的体系结构 …… 204
10.4.4 数据仓库设计 …… 205
10.4.5 数据挖掘 …… 206
10.5 其他新型的数据库系统 …… 209
10.5.1 多媒体数据库系统 …… 209
10.5.2 空间数据库系统 …… 210
10.5.3 模糊数据库系统 …… 211
10.5.4 智能数据库系统 …… 211
10.6 本章小结 …… 213
10.7 习题 …… 213
10.7.1 名词解释 …… 213
10.7.2 简答题 …… 213

参考文献 …… 214

第1章 数据库概述

信息社会，数据库技术的发展已经成为先进信息技术的重要组成部分，绝大多数的计算机应用系统都离不开数据库的支撑。数据库领域有着自身显著的特点，涉及相当多的概念和理论，本章将逐步引出这些概念，使读者对数据库技术有初步的认识和理解。

1.1 计算机数据管理的发展

1.1.1 数据管理

1. 数据

数据库是计算机信息管理的基础，其研究对象是数据。说起数据，人们首先想到的是数字，其实数字只是最简单的一种数据。数据的种类很多，它不仅仅包括数字、字母、文字和其他特殊符号，而且还包括图形、图像、声音等。可以说在日常生活中数据无处不在。

数据是对客观事物的符号表示，通常以一组“数字”组成，是用以表征某一自然客体或社会客体的数量或质量的概念，即数据是用以表征物质的存在、相互关系、运动状态和变化规律的一组“数字”的组合。

计算机处理的数据是经过抽象的，但是数据本身及其解释是分不开的。例如数据 87，这可能是某个人心跳的频率，也可能是某个小孩子的身高，甚至可能是桌子的宽度。所以，离开语义解释，数据就失去了意义。

在日常生活中，人们直接用自然语言描述事物。在计算机中，为了存储和处理这些事物，就要抽出对这些事物感兴趣的特征，组成一个记录来描述它。例如，在图书记录中，如果人们最感兴趣的是图书的书名、作者、出版社、出版时间、价格，则可以这样描述：(数据库原理，张三，清华大学出版社，2004 年 2 月，20.50)。

对于上面这条图书记录，了解其语义的人会得到如下信息：数据库原理是一本书，作者是张三，由清华大学出版社于 2004 年 2 月出版，价格为 20.50 元；而不了解其语义的人则无法理解其含义。可见，数据的形式本身并不能完全表达其内容，需要经过语义解释。数据与其语义是不可分的。

2. 数据与信息

在计算机科学领域，数据是由能被计算机识别与处理的数值、字符等符号构成的集合。

数据只是信息的一种特定的符号表示形式，是计算机程序进行“加工”的原料的总称。

信息有广义和狭义之分。广义的信息是指反映客观事物的特征与变化的资料和消息。狭义的信息是指经过加工、整理，被接收者接收，并对其完成某项业务具有使用价值的情报资料和消息。

数据和信息的含义并不完全相同。数据是指记载下来的事实，是客观实体属性的值，而信息是一种已经被加工为特定形式的数据，是数据所表达的内容，它能使事物的不确定性减少。

数据与信息有着不可分割的联系。一方面，数据是信息的符号表示，或称载体；信息是经过加工的数据，是数据的内涵，是数据的语义解释。另一方面，信息是抽象的，不随数据设备决定的数据形式而改变；而数据的表示方式却具有可选择性。

数据处理是指将数据转换成信息的过程，如对数据的采集、存储、传播、检索、分类、计算等，打印各类报表或输出各种需要的图形。在数据处理的一系列过程中，对数据的采集、存储、传播、检索、分类、计算等操作是基本环节，这些基本环节统称为数据处理。

1.1.2　数据库技术的产生和发展

数据管理是指如何对数据进行分类、组织、编码、存储、检索和维护，它是数据处理的中心问题。而数据库技术正是应数据管理任务的需要而产生的。

随着计算机硬件和软件的发展，数据管理经历了人工管理、文件系统和数据库系统三个发展阶段。

1. 人工管理阶段

在20世纪50年代中期以前，计算机数据管理的能力很差，这一阶段称为人工管理阶段。当时的计算机硬件存储设备只有纸带、卡片、磁带等，没有磁盘等直接存取的存储设备；软件方面没有操作系统，没有管理数据的软件；数据处理方式是批处理；计算机主要用于科学计算。

人工管理数据具有如下特点。

(1) 数据不能长期保存

当时计算机主要用于科学计算，一般不需要将数据长期保存，只是在计算某一课题时将数据输入，用完就撤走。不仅对用户数据如此处置，对系统软件有时也是这样操作。

(2) 没有软件系统负责数据的管理工作

应用程序中不仅要规定数据的逻辑结构，而且要设计物理结构，包括存储结构、存取方法、输入方式等。

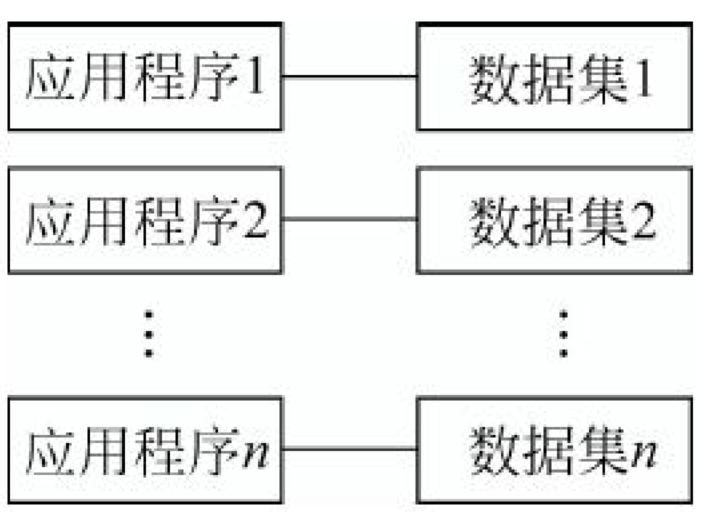

图1-1　人工管理阶段程序与数据的对应关系

(3) 数据不具有独立性

数据的逻辑结构或物理结构发生变化后必须修改应用程序，这就进一步加重了程序员的负担。

(4) 数据不能共享

数据是面向应用的，一组数据只能对应一个程序，如图1-1所示。即使多个应用程序涉及某些相同的数据，也必须各自定义，无法互相利用、互相参照，因此有大量的冗余数据。

2. 文件系统阶段

20 世纪 50 年代后期到 60 年代中期，计算机应用逐渐扩大到管理，计算机数据管理进入到文件系统阶段。这时硬件已有磁盘、磁鼓等直接存取的存储设备；软件方面，操作系统中已经有了专门的数据管理软件，一般称为文件系统；处理方式上不仅有文件批处理，而且能够进行联机实时处理。

用文件系统管理数据具有如下特点。

(1) 数据可以长期保存

由于计算机大量用于数据处理，数据需要长期保留在外存上，反复进行查询、修改、插入和删除等操作，因此数据与程序可以分开存储。

(2) 由专门的软件(即文件系统)进行数据管理

程序和数据之间由软件提供的存取方法进行转换，使应用程序与数据之间有了一定的独立性。数据的存取以记录为基本单位，并出现了多种文件组织，如顺序文件、索引文件和随机文件等。

(3) 数据独立性低

文件系统只实现了数据的物理独立性，而没有实现数据的逻辑独立性。文件系统中的文件是为某一特定应用服务的，文件的逻辑结构对某应用程序来说是优化的，因此要想再增加一些新的应用会很困难，系统不容易扩充，一旦数据的逻辑结构改变，必须修改应用程序，修改文件结构的定义。而应用程序的改变，如改用不同的高级语言等，也将引起文件数据结构的改变，因此数据与程序之间仍缺乏独立性。

(4) 数据共享性差

在文件系统中，一个文件基本上对应于一个应用程序，即文件仍然是面向应用的，如图 1-2 所示。当不同的应用程序具有部分相同的数据时，也必须建立各自的文件，而不能共享相同的数据，因此数据的冗余度大，同时，重复存储也容易造成数据的不一致性。

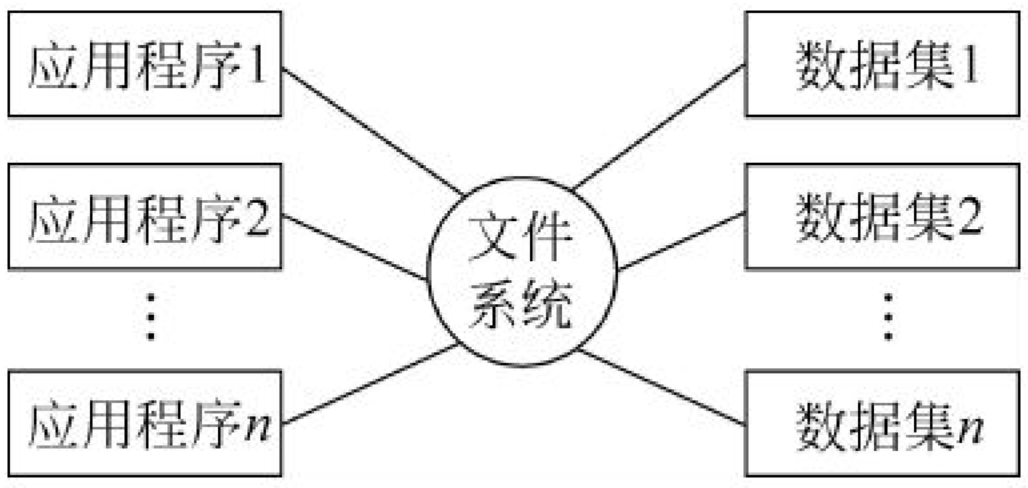

图 1-2　文件管理阶段程序与数据的对应关系

3. 数据库系统阶段

20 世纪 60 年代后期以来，计算机用于管理的规模越来越庞大，应用越来越广泛，数据量急剧增长，同时，多种应用、多种语言互相覆盖地共享数据集合的要求越来越强烈，计算机管理进入到数据库系统阶段。此时硬件已有大容量磁盘，硬件价格下降，软件价格上升，为编制和维护系统软件及应用程序所需的成本相对增加；在处理方式上，联机实时处理的要求更多，并开始提出和考虑分布处理。在这种背景下，出现了数据库技术，由统一的专门软

件系统——数据库管理系统来管理数据。

用数据库系统来管理数据具有如下特点。

(1) 数据结构化

数据结构化是数据库系统与文件系统的根本区别。

在文件系统中,相互独立的文件的记录内部是有结构的。但数据文件之间无联系,数据是面向具体应用的,没有灵活性。而数据库系统在描述数据时不仅描述数据本身,还描述数据之间的联系,能够展现整体数据的结构化。这是数据库的主要特征之一,也是数据库系统与文件系统的本质区别。

在数据库系统中,不仅数据是结构化的,而且存取数据的方式也很灵活,可以存取数据库中的某一个数据项、一组数据项、一个记录或一组记录。而在文件系统中,数据的最小存取单位是记录,其粒度不能细到数据项。

(2) 数据独立性高

数据库系统中的数据既具有物理独立性,又具有逻辑独立性。逻辑独立性保证了当数据的总体逻辑结构改变时,应用程序不必修改;物理独立性保证了当数据的存储结构(即物理结构)改变时,可以保持数据的逻辑结构不变,从而应用程序也不必改变。

数据与程序之间的独立性,使得可以把数据的定义和描述从应用程序中分离出去;大大减少了应用程序的维护和修改工作。

(3) 数据的共享性好,冗余度低

数据的共享程度直接关系到数据的冗余度。数据库系统从整体角度看待和描述数据,数据不再面向某个应用而是面向整个系统,如图 1-3 所示。这样既可以大大减少数据冗余,节约存储空间,又能够避免数据之间的不相容性与不一致性。

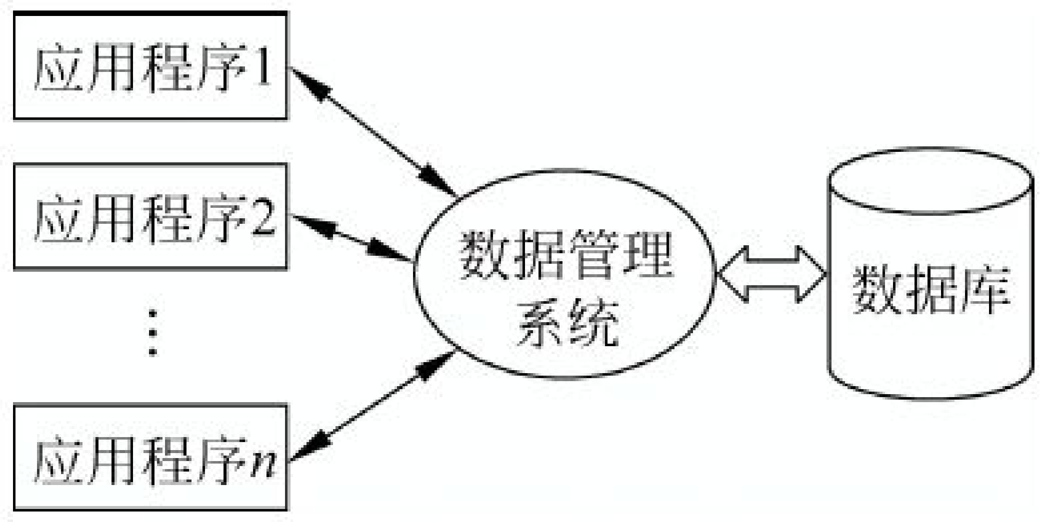

图 1-3 数据库系统阶段程序与数据的对应关系

(4) 数据由数据库管理系统统一管理和控制

由于对数据实行了统一的管理,而且所管理的是有结构的数据,因此在使用数据时可以有很灵活的方式。同时,除了管理功能以外,数据库管理系统还能够保证数据的安全性、完整性,进行并发控制和数据库恢复。

由于数据库面向日常事务处理,不适合进行分析处理,一种新的技术应运而生,这就是数据仓库技术。数据仓库技术是公认的有利于信息利用的最佳解决方案,它不仅能从容解决信息技术人员面临的问题,同时也为商业用户提供了很好的商业契机。数据仓库已成为建立现代决策支持系统的重要技术手段。关于数据仓库的具体内容将在第 10 章详细介绍。

1.2 数据库管理系统

1.2.1 数据库管理系统的定义

数据库,可以直观地理解为存放数据的仓库,只不过这个仓库位于计算机的大容量存储器上,数据不仅要按一定的格式存放,还要便于查找。因此,所谓数据库就是长期存储在计算机内、有组织的、可共享的数据集合。数据库中的数据按一定的数据模型组织、描述和存储,具有较小的冗余度、较高的数据独立性和易扩展性,并可为各种用户共享。

了解了数据库和数据库系统的概念,就应该研究如何利用计算机有效地组织和存储数据、获取和管理数据,完成这个任务的是数据库管理系统。

数据库管理系统(Database Management System,DBMS)是位于用户和操作系统之间的一层数据管理软件,它使用户能方便地定义数据和操纵数据,并能够保证数据的安全性、完整性、多用户并发使用及发生故障后的系统恢复。

1.2.2 数据库管理系统的功能

数据库管理系统是用于建立、使用和维护数据库的一组软件,目前常用的 DBMS 有 SQL Server、Oracle、MySQL、Sybase 等。只有在计算机上配置了 DBMS 之后,才能建立所需要的数据库。

DBMS 的主要功能如下。

1. 数据定义功能

DBMS 提供数据定义语言(Data Definition Language,DDL),用户通过它可以对数据库的结构进行描述,包括模式、外模式、内模式的定义(具体应用时如数据库、基本表、视图、索引),数据库的完整性定义,安全保密定义等。这些定义存储在数据字典中,是 DBMS 运行的基本依据。

2. 数据操纵功能

DBMS 提供数据操纵语言(Data Manipulation Language,DML),实现对数据库中数据的基本操作,如查询、插入、删除和修改。DML 分为两类：宿主型和自含型。所谓宿主型是指将 DML 语句嵌入到某种主语言中使用；自含型是指可以单独使用 DML 语句,供用户交互使用。

3. 数据库的运行管理

这是 DBMS 的核心部分,所有数据库的操作都要在这些控制程序的统一管理下进行。数据库的运行管理包括数据库在运行期间多用户环境下的并发控制(即处理多个用户同时使用某些数据可能产生的问题)、安全性检查、完整性约束条件的检查和执行、运行日志的组织管理和数据库内部维护等。

4. 数据库的建立和维护功能

数据库的建立和维护是DBMS的一个重要组成部分，它包括数据库初始数据的输入、数据转换，数据库的转储、恢复功能，数据库的重新组织功能和性能监视、分析功能等，这些功能通常是由一些实用程序完成的。

5. 数据组织、存储和管理

数据库中需要存放多种数据，如数据字典、用户数据、存取路径等。DBMS负责分门别类地组织、存储和管理这些数据，确定以何种文件结构和存取方式物理地组织这些数据，如何实现数据之间的联系，以便提高存储空间利用率以及提高随机查找、顺序查找、增、删、改等操作的效率。

6. 数据通信功能

DBMS提供与其他软件系统进行通信的功能，例如与操作系统的联机处理、分时处理和远程作业传输的相应接口，又如与其他DBMS或文件系统的接口，从而能够将该DBMS下的数据转换为其他DBMS或文件系统能够识别的数据或者能够接收另一个DBMS或文件系统的数据。

1.3 数据库系统

1.3.1 数据库系统的定义

数据库本身不是独立存在的，它是数据库系统的一部分，在实际应用中，人们所面对的数据库系统(Database System，DBS)是指带有数据库的计算机应用系统。

1.3.2 数据库系统的组成

数据库系统是计算机应用系统中引入数据库后的系统。简单地讲，是由硬件、软件、数据库和人员组成的：具体来讲，一般由硬件系统、数据库管理系统及相关软件、数据库集合和人员组成(如图1-4所示)。

硬件系统是整个数据库系统的基础，需要有足够大的内存、足够大容量的磁盘等联机直接存取设备等；数据库管理系统是管理数据库的软件，实现数据库的建立、使用和维护等功能，它是数据库系统的核心，相关软件是支持软件(如操作系统、语言编译系统、应用程序等)；数据库集合是相关数据的集合，是若干个设计合理、满足应用需要的数据库；人员主要包括数据库管理员、系统分析员、程序员和用户，其中数据库管理员是对DBMS直接进行操作的人员，其作用非常重要。

数据库管理员(Database Administrator，DBA)是一个或一组人员，是全面负责数据库的建立、维护和管理数据库系统的人员，其具体的职责包括：

- 设计与定义数据库系统。数据库中要存放哪些数据，是由系统需求来决定的。为了

更好地对数据库系统进行有效的管理和维护，DBA 应该参加、了解数据库建设的全过程，并与系统分析员、程序员和用户共同协商，搞好数据库设计，并决定数据库的存储结构和存取策略，以获得较高的存取效率和存储空间利用率。

- 监控数据库的使用和运行。DBA 负责监视数据库系统的运行情况，及时处理运行过程中出现的问题，定义数据的安全性要求和完整性约束条件，收集系统审计信息。
- 数据库的转储与恢复。当数据库系统发生故障时，数据库中的数据可能会遭到不同程度的破坏，DBA 必须在最短的时间内将数据库中的数据恢复到某种一致状态。为此，DBA 要定义和实施适当的后援和恢复策略，如周期性地转储数据和维护日志文件等。
- 数据库的改进和重组。DBA 还负责监视和分析系统运行期间的性能（如处理效率），对运行情况进行记录、统计分析，依靠工作实践并根据实际应用环境不断改进数据库设计。同时，在数据库运行过程中，大量数据不断被增、删、改，随着运行时间的延长，可能会影响系统的性能，因此需要 DBA 定期对数据库进行重新组织。
- 数据库的重构。当用户的需求增加或改变时，DBA 还要对数据库进行较大的改造，包括增加和修改部分设计。

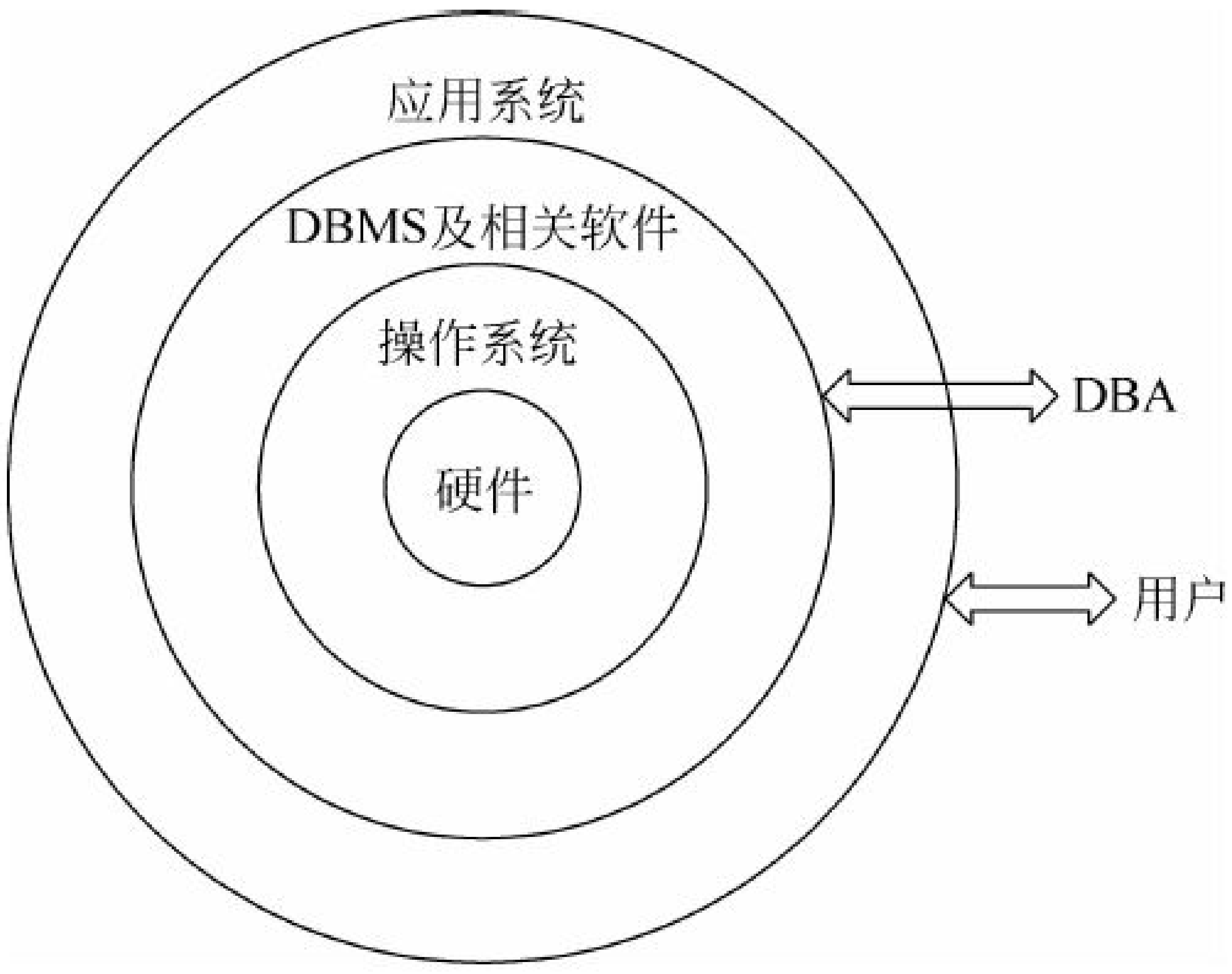

图 1-4 数据库系统的组成

1.3.3 数据库系统的模式

当设计数据库时，人们对数据库的结构感兴趣；当应用数据库时，人们关心的是数据库中的数据。数据库中的数据经常变化，而数据库的结构在一定时间范围内不会改变。

在数据模型中有“型”和“值”的概念：型是指对某一类数据的结构和属性的说明，即数据库的结构；值是型的一个具体赋值，即数据库中的具体数据。

数据库中结构的定义可以在多个抽象级别上进行，形成多个级别的数据库模式。模式是数据库中全体数据的逻辑结构和特征的描述，它仅仅涉及型的描述，不涉及具体的值。模式的一个具体值称为模式的一个实例。同一个模式可以有很多个实例。模式是相对稳定的，而实例是相对变动的。模式反映的是数据的结构及其关系，而实例反映的是数据库在某一时刻的状态。

1. 数据库的三级模式结构

图 1-5 说明了数据库系统的三级模式：外模式、模式和内模式。

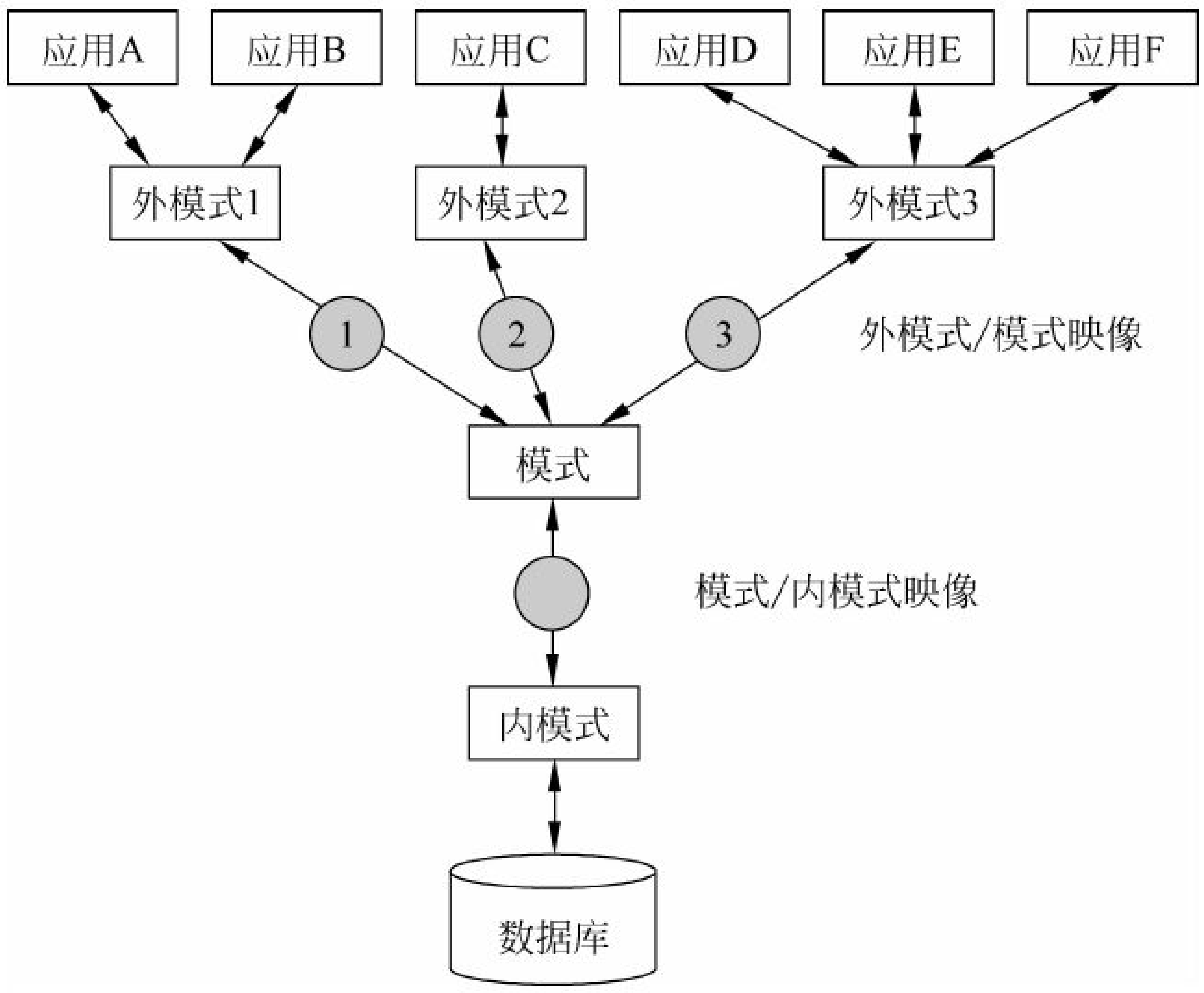

图 1-5　数据库系统模式结构

（1）模式

模式也称逻辑模式或概念模式，是对数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图，它是数据库系统模式结构的中间层，不涉及数据的物理存储细节和硬件环境，与具体的应用程序，以及所使用的应用开发工具和高级程序设计语言无关。

实际上模式是数据库数据在逻辑级上的视图。一个数据库只有一个模式。定义模式时不仅要定义数据的逻辑结构，而且要定义与数据有关的安全性、完整性要求，定义这些数据之间的联系。

（2）外模式

外模式也称子模式或用户模式，它是对数据库用户（包括应用程序员和最终用户）看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。

外模式通常是模式的子集。一个数据库可以有多个外模式。即使是模式中同一数据，在外模式中的结构、类型、长度、保密级别等都可以不同。另一方面，同一外模式也可以为某一用户的多个应用系统所使用，但一个应用程序只能使用一个外模式。

外模式是保护数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据，数据库中的其余数据对他们来说是不可见的。

（3）内模式

内模式也称存储模式，它是对数据物理结构和存储结构的描述，是数据在数据库内部的表示方式。一个数据库只有一个内模式。

图 1-6 是关于三级模式的示例。

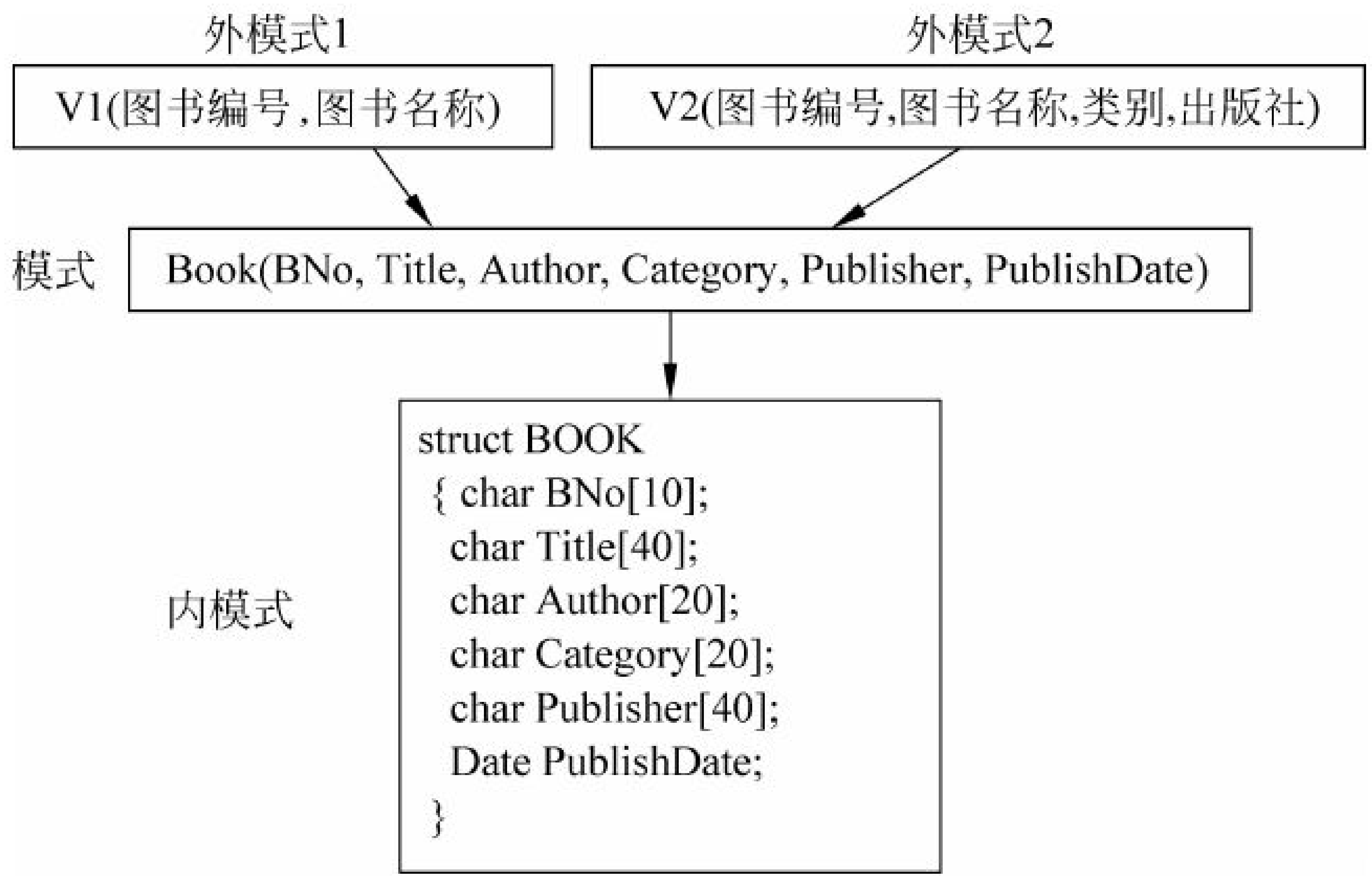

图 1-6 数据库三级模式示例

2. 数据库的两级映像功能

DBMS 为实现三级模式结构,不仅提供了定义内模式、模式、外模式的语言,而且还在三级模式之间提供了两级映像:外模式/模式映像和模式/内模式映像。正是这两级映像保证了数据库系统中的数据具有较高的逻辑独立性和物理独立性。

(1) 外模式/模式映像

模式描述的是数据的全局逻辑结构,外模式描述的是数据的局部逻辑结构对应于同一个模式可以有任意多个外模式。对于每一个外模式,数据库系统都有一个外模式/模式映像,它定义了该外模式与模式之间的对应关系。这些映像定义通常包含在各自外模式的描述中。当模式改变时(如增加新的数据类型、新的数据项、新的关系等),由数据库管理员对数据库外模式/模式映像做相应的改变,可以使外模式保持不变。从而不必修改应用程序,保证了数据的逻辑独立性。

(2) 模式/内模式映像

数据库中只有一个模式,也只有一个内模式,所以模式/内模式映像是唯一的,它定义了数据全局逻辑结构与存储结构之间的对应关系。例如,说明逻辑记录和字段在数据库内部是如何表示的。该映像定义通常包含在模式描述中。当数据库的存储结构发生改变时(例如,采用了更先进的存储结构),由数据库管理员对模式/内模式映像做相应的改变,可以使模式保持不变,从而保证了数据的物理独立性。

数据独立性是数据库系统追求的目标。DBMS 实现的三级模式和二级映像机制,使得数据库系统具有较强的逻辑独立性和物理独立性。因为特定的应用程序是在外模式描述的数据结构的基础上编写的,依赖于特定的外模式,但独立于数据库的模式和存储结构。数据库的二级映像保证了数据库外模式的稳定性,从而从底层保证了应用程序的稳定性。另一方面,数据与应用程序的独立性,使得数据的定义和描述可以从应用程序中分离出去。而且,由于数据的存取和存储由 DBMS 管理,用户不必考虑存取路径等细节,因此简化了应用程序的编制,减少了应用程序的维护与修改工作,提高了应用程序的质量。

1.3.4 数据库语言

数据库语言包括数据描述语言、数据操纵语言、数据控制语言和宿主语言。

1. 数据描述语言

数据描述语言(DDL)用于描述数据库中各种对象的特征,主要描述数据的逻辑结构、数据的物理特征、逻辑数据到物理数据的映射(通常称为存储映射)和访问规则(如用户与外模式的对应关系、用户身份确认等)。

数据库三级模式均使用 DDL 来描述,只是内模式由内模式 DDL 定义,而物理数据库的详细设计由 DBMS 的设备介质语言来定义。

2. 数据操纵语言

数据操纵语言(DML),是用户与数据库系统的接口之一,是用户操作数据库中数据的工具。使用 DML 可以实现数据库数据的插入、删除、修改、查询、统计等操作。在设计 DML 时,一般要做到描述操作准确,无二义性;功能齐全,操作能力强。用户希望使用的操作应尽量满足语言自然、直观,容易掌握,使用方便等要求。

3. 数据控制语言

数据控制语言(Data Control Language,DCL),是数据库中实现数据访问权限控制以及事务管理的一类语言。例如,限制用户对某类数据的插入、删除、修改、查询等操作,或取消一些限制,当数据库操作完成时提交事务或回滚事务等。对数据库对象进行操作的授权工作一般由数据对象创建者或数据库管理员完成。

4. 宿主语言

宿主语言是通常的程序设计语言(如 C 语言)。数据库上的操作常用应用程序实现,而且应用程序还要完成许多非数据库上的操作,因此操作数据库的程序一般用宿主语言写成。

宿主语言可以通过两种方式引用 DML 的命令:

- 通过 DBMS 提供的过程(或函数)引用。
- 将 DML 嵌入宿主语言中,与宿主语言一起使用。

第二种方式可能有一个预编译器处理 DML,或有一个编译器既能处理宿主语言又能处理 DML,并将 DML 语句转换成调用 DBMS 提供的过程(或函数)。

1.4 数据模型

数据库中的数据是有结构的,即不仅包含数据本身的内容,还包含数据之间的关系。因为计算机不可能直接处理现实世界中的具体事物,所以必须事先把具体事物转换成计算机能够处理的数据。在数据库中,是利用数据模型这个工具来抽象、表示和处理现实世界中的数据和信息的。

1.4.1　数据处理的三个领域

数据从现实世界到数据库里的具体表示要经历三个领域，即现实世界、信息世界、计算机世界。这三个领域的关系如图 1-7 所示。

1. 现实世界

现实世界是指客观存在世界中的事物及联系，是存在于人们头脑之外的客观世界。现实世界的数据就是客观存在的各种报表、图标和查询格式等原始数据。计算机要能处理数据，首先要解决的问题就是按用户的观点对数据和信息建模，即抽取数据库技术所研究的数据，然后分门别类，综合出系统所需要的数据。

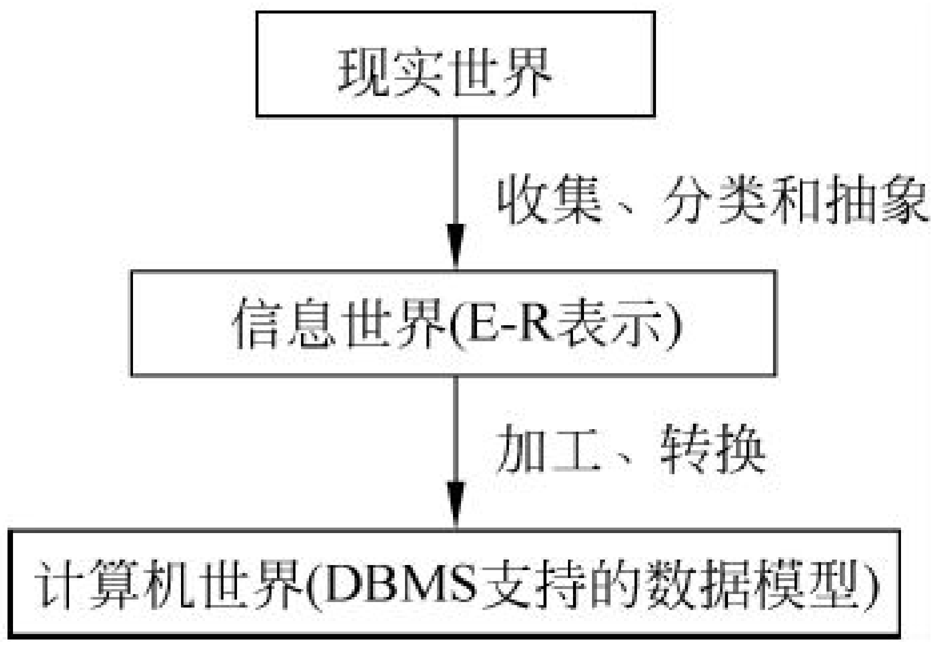

图 1-7　数据处理的三个领域

2. 信息世界

信息世界（或称概念世界）是现实世界在人们头脑中的反映，是对客观事物及其联系的一种抽象描述。人们用符号、文字将数据记录下来。在信息世界中，一般使用实体-联系（Entity-Relationship，E-R）图表示概念数据模型。

信息世界涉及以下主要概念。

（1）实体

实体（Entity）是现实世界中客观存在并可相互区分的事件或物体。例如，一本书、一个学生、一间教室等都是实体。实体可以是具体的人、事、物，也可以是抽象的概念或联系。

同类型实体的集合称为实体集。

（2）属性

实体所具有的某一特性称为属性（Attribute）。实体都是由一组属性来表示的，如某网站会员可以用（用户名，密码，性别，出生日期…）来描述。属性的具体取值称为属性值，用以表示一个具体的实体。

（3）码

可以唯一标识实体的属性称为码（Key）。例如，图书的码为图书编号。

（4）域

属性的取值范围称为该属性的域（Domain）。例如，图书名称和作者的域为字符串集合，会员性别的域为（男，女），出生日期的域为日期。

（5）实体集

实体集（Entity Set）是具有相同属性的实体集合。例如，图书馆中所有供借阅的图书就是一个图书的实体集。

（6）实体型

实体型（Entity Type）是对实体集的抽象描述，用实体名及其属性名集合来抽象和刻画同类实体，例如图书（编号、书名、作者、类别、出版社、出版时间）。

（7）联系

在现实世界中，事物内部以及事物之间是有联系的，这些联系在信息世界中反映为实体

内部的联系(Relationship)和实体之间的联系。实体内部的联系反映数据在同一个实体内各属性之间的联系。

① 两个实体之间的联系

• 一对一联系(1∶1)

如果对于实体集 A 中的每一个实体,实体集 B 中至多有一个实体与之联系,反之亦然,则称实体集 A 与实体集 B 具有一对一联系,记为 1∶1。

例如,公司里面,一个部门只有一个经理,而一个经理只在一个部门中任职,则部门集与经理集之间的联系为 1∶1。

• 一对多联系(1∶n)

如果对于实体集 A 中的每一个实体,实体集 B 中有 $n(n\geqslant 0)$ 个实体与之联系;对于实体集 B 中的每一个实体,实体集 A 中至多有一个实体与之联系,则称实体集 A 与实体集 B 具有一对多联系,记为 1∶n。

例如,一个部门有多个职工,而一个职工只在一个部门工作,则部门集与职工集之间的联系为 1∶n。

• 多对多联系(m∶n)

如果对于实体集 A 中的每一个实体,实体集 B 中有 $n(n\geqslant 0)$ 个实体与之联系;对于实体集 B 中的每一个实体,实体集 A 中有 $m(m\geqslant 0)$ 个实体与之联系,则称实体集 A 与实体集 B 具有多对多联系,记为 m∶n。

例如,一个产品可以由多个零件组成,而一个零件又可以用于多种产品,则产品集与零件集之间的联系为 m∶n。

对信息世界建模,必须方便、准确地表示出上述信息世界中的常用概念,最常用的工具是实体-联系图(E-R 图)。E-R 图提供以下描述实体型、属性和联系的方法(如图 1-8 所示)。

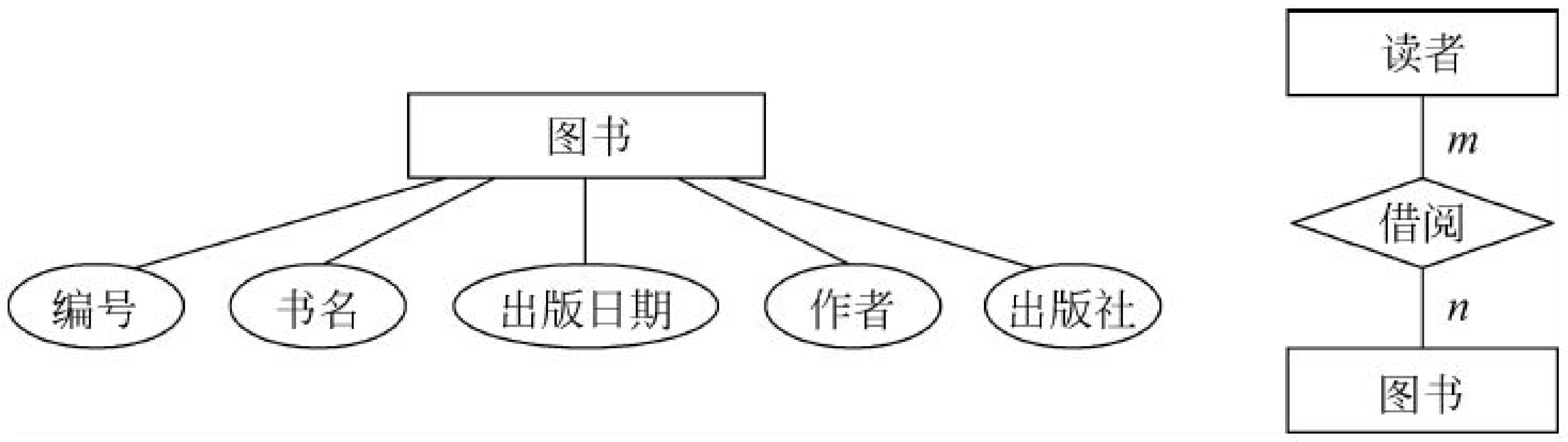

图 1-8 E-R 图示例

实体型:用矩形表示,矩形框内写明实体名。

属性:用椭圆形表示,并用无向边将其与相应的实体连接起来。

联系:用菱形表示,菱形框内写明联系名,并用无向边分别与有关实体型连接起来,同时在无向边旁标上联系的类型。注意:联系本身也是一种实体型,也可以有属性,联系的属性也要用无向边与该联系连接起来。

两个实体间的三种联系,可以用图 1-9 表示。

当实体及联系较多时,可以将 E-R 图分成两部分:实体及属性图和实体及联系图(如图 1-10 所示)。

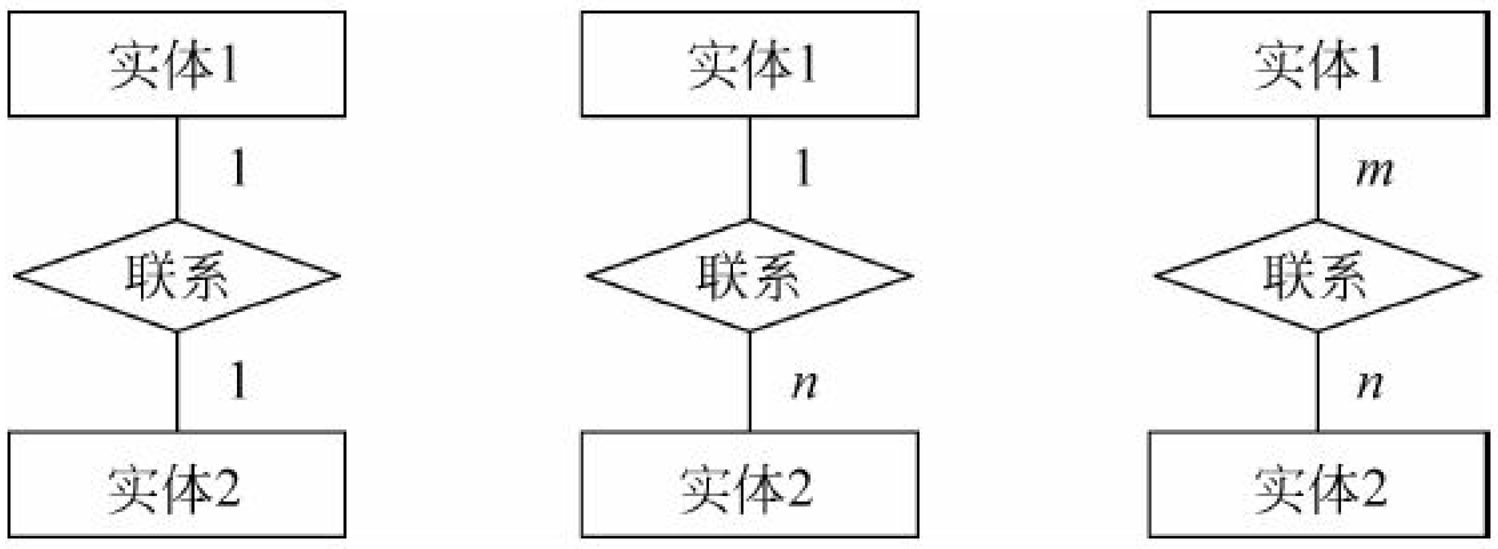

图 1-9 两个实体间的三种联系

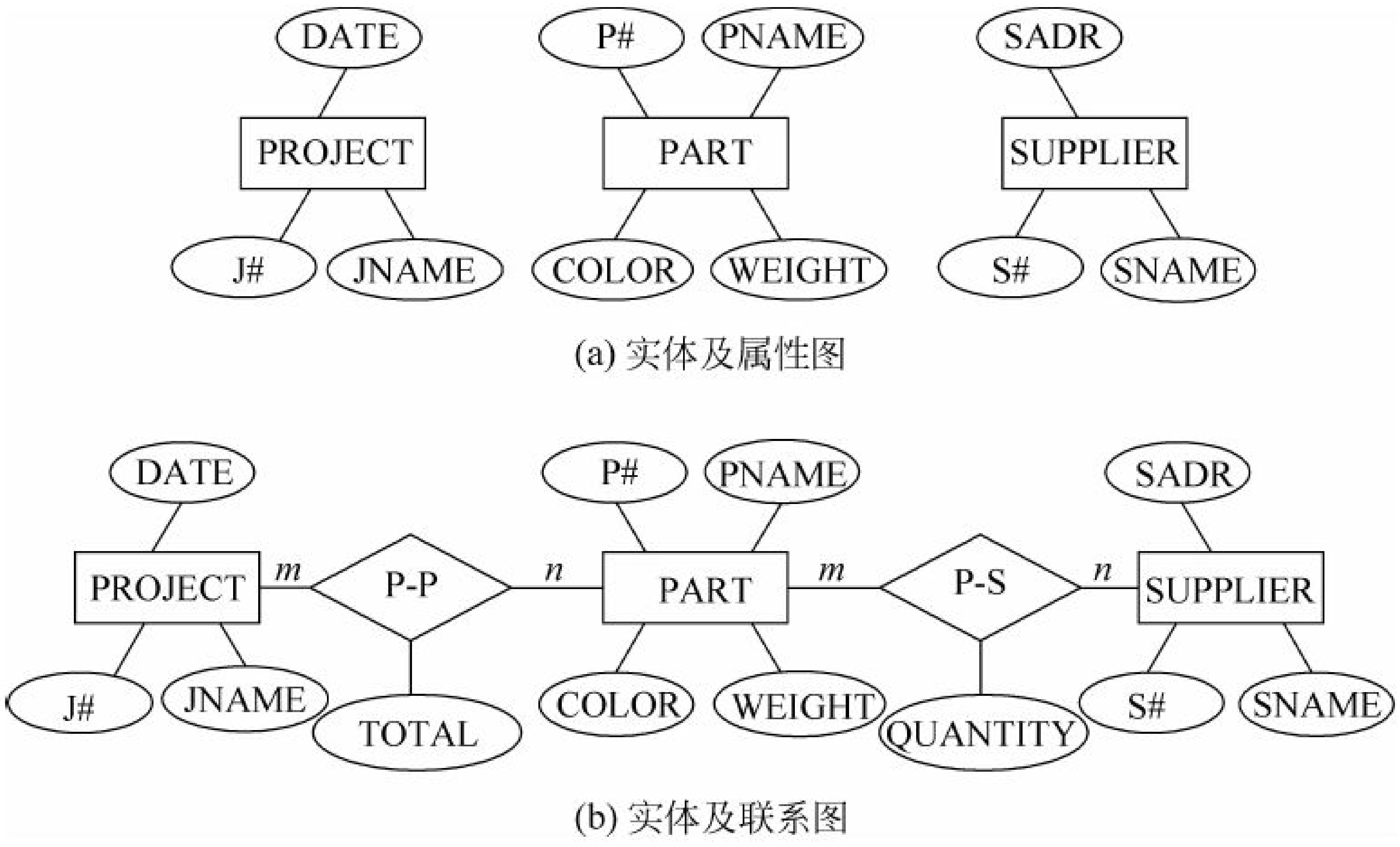

图 1-10 E-R 图示例

② 一个实体内的联系

一个实体内的联系也包括 1∶1、1∶n、m∶n 三种。例如,学生实体型内部具有领导和被领导的联系,即班长领导班里的学生,而每个学生只被一个班长领导,如图 1-11 所示。

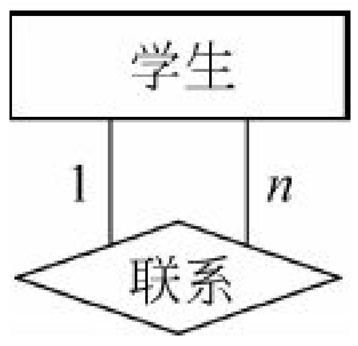

图 1-11 一个实体内的联系示例

③ 多个实体间的联系

多个实体间的联系也包括 1∶1、1∶n、m∶n 三种。例如,教师、课程和学生三个实体集之间存在以下关系:每个教师可以讲授多门课程给不同的学生,每门课程可能会有多个教师讲授,每个学生可以上多门课程,每门课程都有很多学生选修,所以他们之间是多对多的联系。又如,每个保管员工作在一个仓库中,保管着多种商品,每种商品只能存放在一个仓库中,每个仓库由多名保管员看管。以上两个例子中多个实体间的联系如图 1-12 所示。

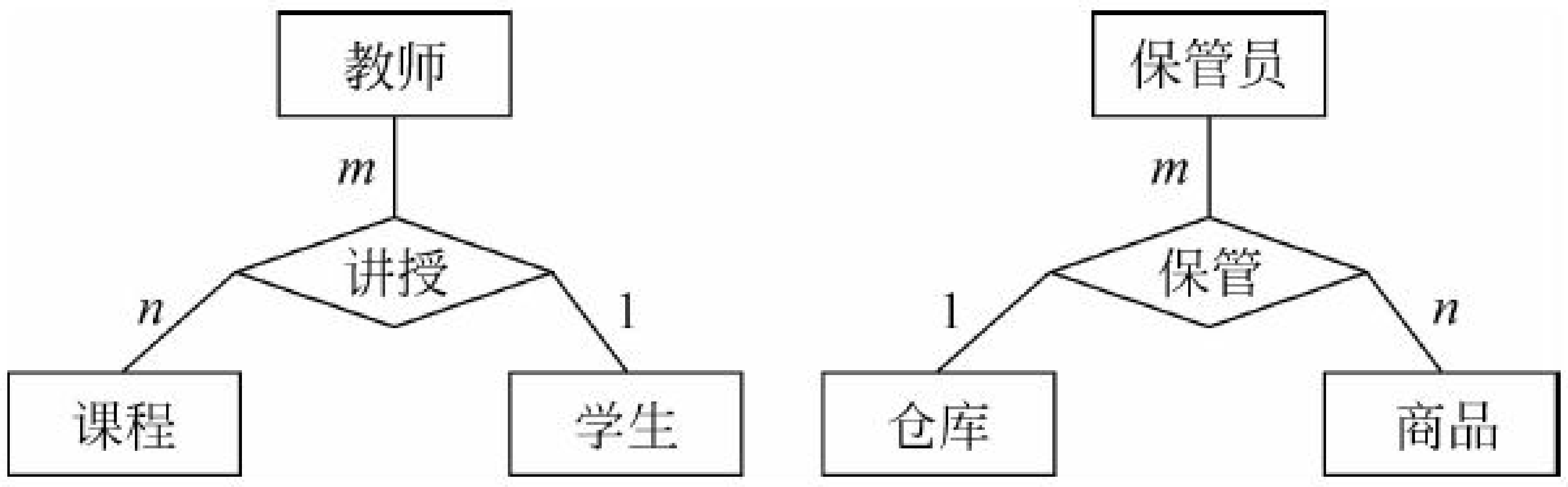

图 1-12 三个实体间的联系示例

3. 计算机世界

计算机世界也可称做数据世界或机器世界，是在信息世界基础上的进一步抽象，表现为DBMS支持的数据模型。

1.4.2 数据模型的要素

数据模型是对客观事物及其联系的数据描述，需要描述系统的静态特性、动态特性和约束条件。

1. 数据结构

数据结构用于描述系统的静态特性。

数据结构是所研究的对象类型的集合。这些对象是数据库的组成成分，包括两类：一类是与数据类型、内容、性质有关的对象；另一类是与数据之间联系有关的对象。

数据结构是刻画一个数据模型性质最重要的方面。因此在数据库系统中，通常按照其数据结构的类型来命名数据模型。

2. 数据操作

数据操作用于描述系统的动态特性。

数据操作是指对数据库中各种对象的实例允许执行的操作的集合，包括操作及有关的操作规则。数据库主要有检索和更新(插入、删除、修改)两大类操作。数据模型必须定义这些操作的确切含义、操作符号、操作规则(如优先级)以及实现操作的语言。

3. 数据的约束条件

数据的约束条件是一组完整性规则的集合；完整性规则是给定的数据模型中数据及其联系所具有的制约和存储规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效和相容。

数据模型应该反映和规定数据模型必须遵守的基本的通用的完整性约束条件，此外还应该提供定义完整性约束条件的机制，以反映具体应用所涉及的数据。

1.4.3 数据模型的分类

按照数据模型描述角度的不同，数据模型可以分为以下几类。

- 概念模型：也称信息模型，按用户的观点对数据和信息进行建模，是现实世界到信息世界的第一层抽象，强调其语义表达功能，易于用户理解，是用户和数据库设计人员交流的语言，主要用于数据库设计。在这类模型中，最著名的工具是实体联系模型(E-R模型)，前面已详细介绍。
- 数据库支持的数据模型：它是按计算机系统的观点对数据进行建模，是现实世界数据特征的抽象，用于实现DBMS。不同的数据模型具有不同的数据结构形式，目前最常用的数据模型有层次模型、网状模型和关系模型。随着技术发展，面向对象模型越来越受到关注。

层次模型和网状模型统称为非关系模型。在非关系模型中，实体用记录表示，实体之间的联系转换成记录之间的两两联系。非关系模型数据结构的基本单位是基本层次联系。所谓基本层次联系是指两个记录以及它们之间的一对多(包括一对一)的联系，如图 1-13 所示。图 1-13 中，R_i 位于联系 L_{ij} 的始点，称为双亲结点；R_j 位于联系 L_{ij} 的终点，称为子女结点。

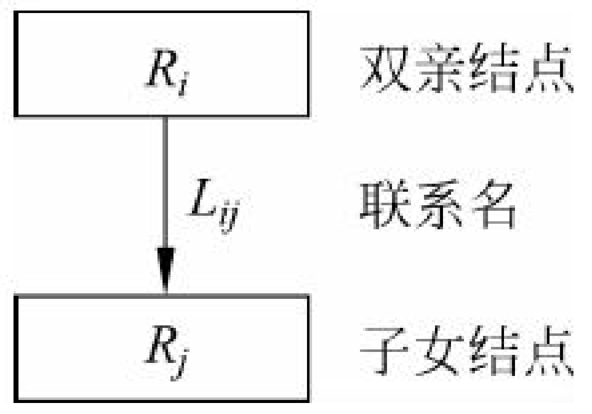

图 1-13 基本层次联系

这三种模型的根本区别在于数据之间联系的方式(即记录型之间的联系方式)不同。关系模型是用“二维表”来表示数据之间的关系；层次模型是用“树结构”来表示数据之间的关系；网状模型是用“图结构”来表示数据之间的关系。由于它们的数据表示方式不同，当用户使用数据库时，关系模型只用了数据记录的内容，使得用户在关系 DBMS 中操作时，不必去了解数据记录的联系及顺序，自然就觉得使用起来简单方便；而层次模型和网状模型要用记录与记录之间的联系，以及它们在存储结构中的具体安排，这就要求用户有较多的计算机知识，对一般用户来说，使用起来就不太简单方便了。下面对这三种模型做一个简单的介绍。

1. 层次模型

层次模型(Hierarchical Model)是数据库系统中最早出现的数据模型，它的数据结构是一棵“有向树”，即采用树状结构表示数据与数据之间的联系。层次模型的特征是：

- 有且仅有一个结点没有父结点，它就是根结点；
- 其他结点有且仅有一个父结点。

在层次模型中，每个结点表示一个实体型，称为记录型。一个记录型可有许多记录值，简称为记录。结点之间的有向边表示记录之间的联系。如果要存取某一记录型的记录，可以从根结点开始，按照有向树层次逐层向下查找，查找路径就是存取路径。

图 1-14 给出了某学校的系、教研室、教师、班级的层次模型及实例。图 1-14(a)中的“系”是根结点，该树状结构反映的是实体型之间的结构。该模型实际存储的数据通过链接

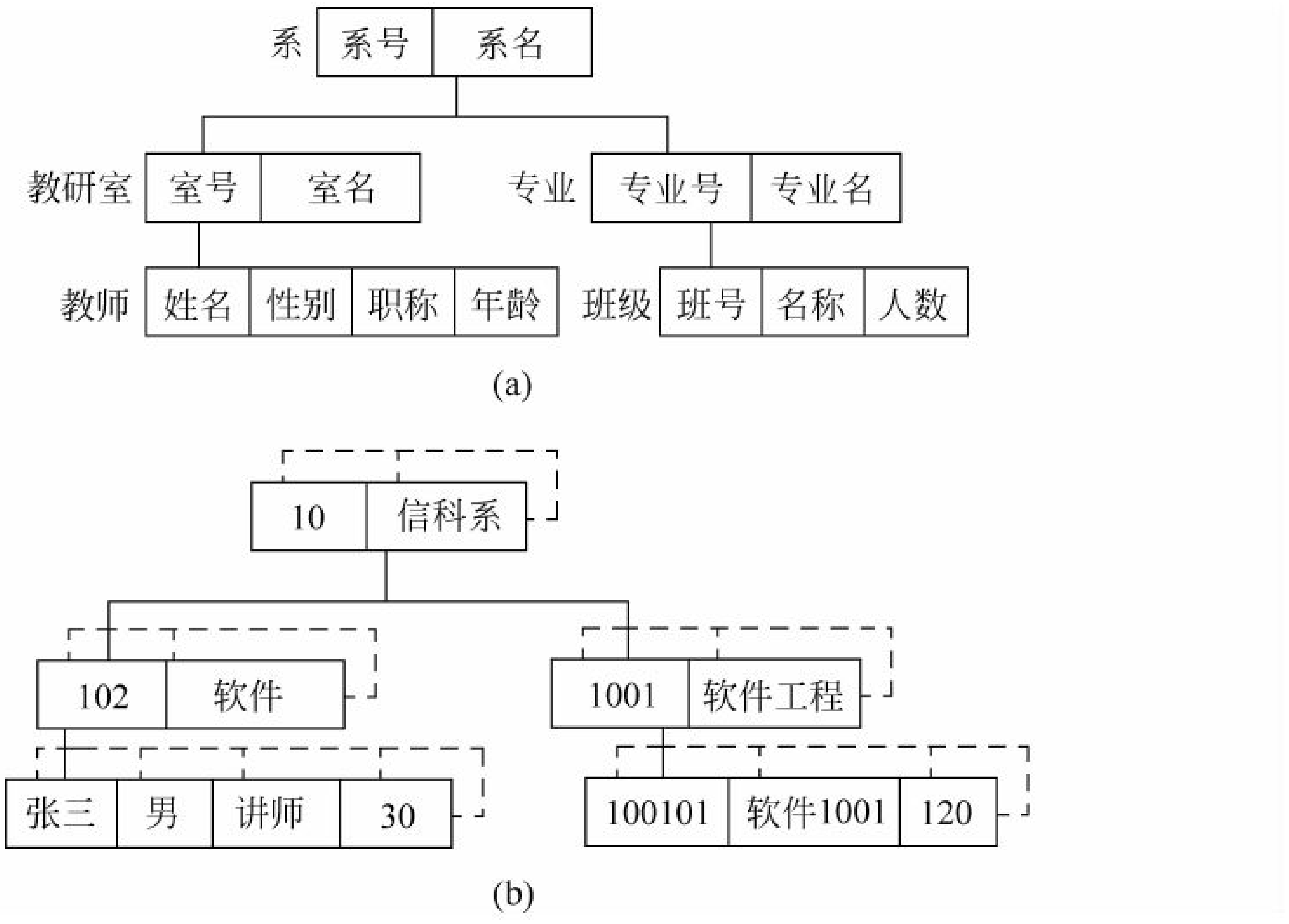

图 1-14 层次模型示意图

指针体现它们之间的这种联系。图 1-14(b)的实例给出了学校里一个系的数据,其他系的情况用虚线框表示,并予以省略。

层次模型不能直接表示多对多的联系。若要表示多对多的联系,可采用如下两种方法。

方法 1:冗余结点法。将两个实体的多对多的联系转换为两个一对多的联系。

方法 2:虚拟结点分解法。将冗余结点转换为虚拟结点。虚拟结点是一个指引元,指向所代替的结点。该方法的优点是减少对储存空间的浪费,避免数据的不一致性;该方法的缺点是改变储存位置有可能引起虚拟结点中指针的修改。

层次模型的特点是记录之间的联系通过指针实现,比较简单,查询效率高。

层次模型的缺点是只能表示 $1:n$ 的联系,尽管有许多辅助手段可以实现 $m:n$ 的联系,但较复杂,不易掌握,由于层次顺序严格、复杂,插入删除操作的限制比较多,导致应用程序编制起来比较复杂。

2. 网状模型

用网状结构表示实体及其之间联系的模型称为网状模型(Network Model)。网状模型(也称 DBTG 模型)是一个比层次模型更具普遍性的数据结构,是层次模型的一个特例。网状模型可以直接描述现实世界,这是因为去掉了层次模型的两个限制,网状模型的特征是:

- 可以有一个以上的结点没有父结点;
- 网中的每一个结点代表一个记录类型,联系用链接指针来实现。

广义地讲,任何一个连通的基本层次联系的集合都是网状模型。

图 1-15 给出了一个简单的网状模型。由于每一个基本层次联系都代表一对多的联系,因此,若将图形倒置也不可能变成层次模型。图 1-15(a)中的每一个结点为一个记录类型,图 1-15(b)是图 1-15(a)的一个具体示例。其中用单向环形链接指针实现联系。可以看出,如果零件和配件数量较多,链接将非常复杂。

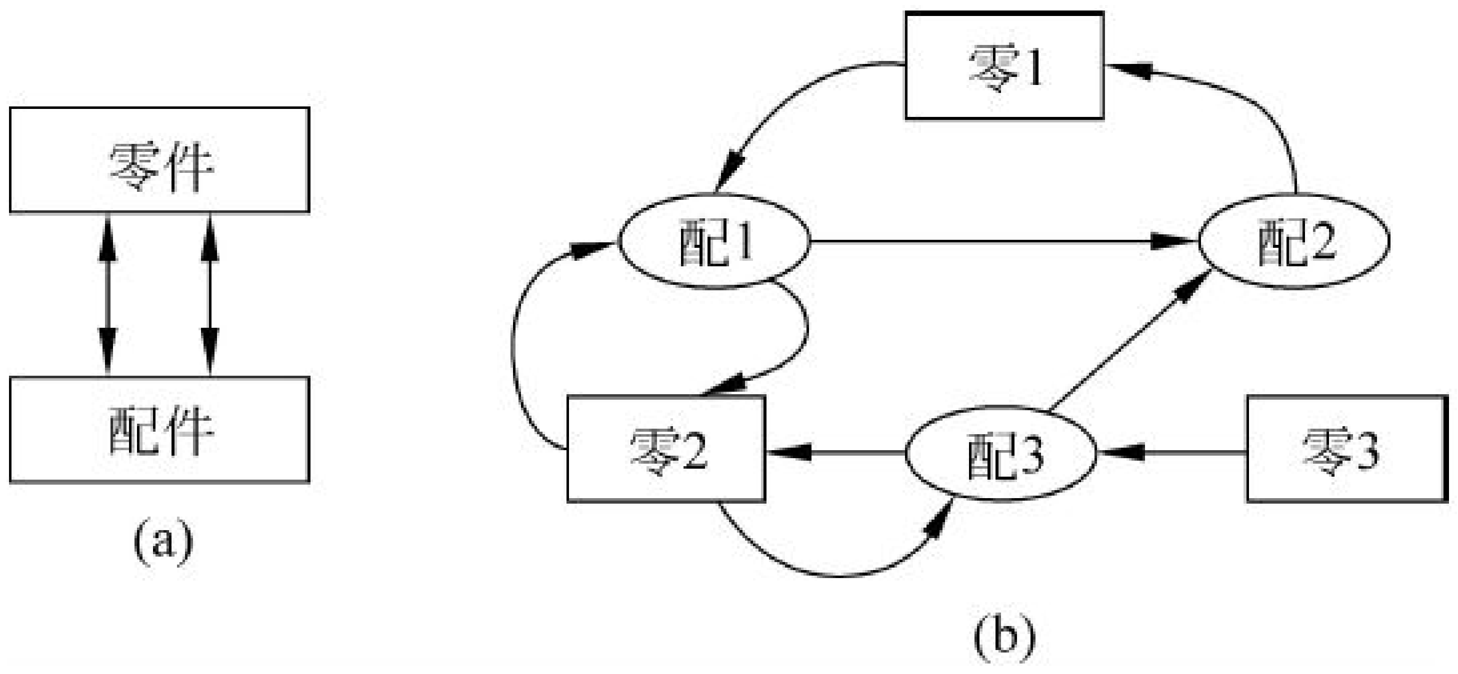

图 1-15 网状模型示意图

网状模型中的每个结点表示一个记录类型(实体),每个记录类型可以包含若干个字段(实体的属性),结点间的连线表示记录类型之间一对多的联系,层次模型和网状模型的主要区别如下:

(1) 网状模型中,子女结点与双亲结点的联系不唯一,因此需要为每个联系命名;

(2) 网状模型允许复合链,即两个结点之间有两种以上的联系;

(3) 网状模型不能表示记录之间的多对多联系,需要引进连接记录来表示多对多的联系。

通常，网状数据模型没有层次模型那样严格的完整性约束条件，但 DBTG 在模型 DDL 中提供了定义 DBTG 数据库完整性的若干概念和语句：

(1) 支持记录码的概念，码能唯一标识记录的数据项集合；

(2) 保证一个联系中双亲记录和子女记录之间是一对多的联系；

(3) 支持双亲记录和子女记录之间的某些约束条件。

网状模型的主要优点是能更为直接地描述现实世界，具有良好的性能，存取效率高。

网状模型的主要缺点是结构复杂。例如，当应用环境不断扩大时，数据库结构就变得很复杂，不利于最终用户掌握，编制应用程序的难度比较大，DBTG 模型的 DDL、DML 语言复杂，记录之间的联系是通过存取路径来实现的，因此程序员必须了解系统结构的细节，增大了编写应用程序的难度。

网状模型和层次模型在本质上是一样的。从逻辑上看，它们都是基本层次联系的集合，用结点表示实体，用有向边表示实体间的联系。从物理结构上看，它们的每一个结点都是一个存储记录，用链接指针来实现实体之间的联系。当存储数据时这些指针就固定下来了，检索数据时必须考虑存取路径问题；数据更新时，涉及链接指针的调整，缺乏灵活性；系统扩充相当麻烦。网状模型中的指针更多，纵横交错，从而使数据结构更加复杂。

3. 关系模型

关系模型是用二维表格结构来表示实体以及实体之间联系的数据模型。关系模型的数据结构是一个“二维表框架”组成的集合，每个二维表又可称为关系，因此可以说，关系模型是“关系框架”组成的集合。目前大多数 DBMS 都是基于关系模型的，所以它是我们重点讨论的数据模型。

关系模型的特征是：

- 描述的一致性，不仅实体用关系描述，实体之间的联系也用关系描述；
- 可用关系直接表示多对多的联系；
- 关系必须是规范化的，即每个属性是不可分的数据项，不允许表中有表。
- 关系模型是建立在数学概念基础上的，有较强的理论基础。

图 1-16 给出了一个简单的关系模型，其中图 1-16(a)给出了关系模式：

部门关系框架

部门编号	名称	经理	人数

员工关系框架

员工编号	姓名	性别	部门编号

(a)

部门关系框架

部门编号	名称	经理	人数
01	人事	A001	8
02	销售	B001	56

员工关系框架

员工编号	姓名	性别	部门编号
A001	张三	男	01
B003	李四	女	02

(b)

图 1-16　关系模型示意图

部门(部门编号,名称,经理,人数)。
员工(员工编号,姓名,性别,部门编号)。

图 1-16(b)给出了这两个关系模式的关系,关系名分别为部门关系和员工关系,均包含两个元组,部门关系的关键字为部门编号,员工关系的关键字为员工编号。

在关系模型中基本数据结构就是二维表,不用像层次模型或网状模型那样的链接指针。记录之间的联系是通过不同关系中的同名属性来体现的。例如,要查找员工“张三”所属的部门,首先要在员工关系中根据姓名找到所属部门 01,然后在部门关系中找到 01 部门编号对应的名称即可。在上述查询过程中,同名属性的“部门编号”起到了联系两个关系的纽带作用。由此可见,关系模型中的各个关系模式不应当孤立起来看待。

4. 面向对象数据模型

面向对象数据模型(Object-Oriented Data Model,简称 OO 模型)是面向对象程序设计方法与数据库技术相结合的产物,用以支持非传统应用领域对数据模型提出的新需求。OO 模型的基本目标是以更接近人类思维的方式描述客观世界的事物及其联系,并且使描述问题的问题空间和解决问题的方法空间在结构上尽可能一致,以便对客观实体进行结构模拟和行为模拟。

在面向对象数据模型中,基本结构是对象(Object)而不是记录,一切事物、概念都可以看做对象。一个对象不仅包括描述它的数据,而且还包括对它进行操作的方法的定义。另外,面向对象数据模型是一种可扩充的数据模型,用户可根据应用需要定义新的数据类型及相应的约束和操作,而且比传统数据模型具有更丰富的语义。

根据数据模型的三要素:数据结构、数据操作和数据约束条件,将面向对象数据模型与关系数据模型做一简单比较:

(1) 在关系数据模型中基本数据结构是表,相当于面向对象数据模型中的类;关系模型中的数据元组相当于面向对象数据模型中的实例(对象),它们的区别在于 OO 模型的类中还包括方法,关系数据模型中只有实体的属性而没有对实体的操作。

(2) 在关系数据模型中,对数据库的操作都归结为对关系的运算,而在面向对象数据模型中对类的操作分为两部分:一部分是封装在类内的操作即方法,另一部分是类间相互沟通的操作即消息。

(3) 在关系数据模型中有域、实体和参照完整性约束,完整性约束条件可以用逻辑公式表示,称为完整性约束方法。在面向对象数据模型中,这些用于约束的公式可以用方法或消息表示,称为完整性约束消息。

总之,面向对象数据模型是用面向对象的观点来描述现实世界实体(对象)的逻辑组织、对象间限制和联系等的模型。它能完整地描述现实世界的数据结构,具有丰富的表达能力,但该模型相对比较复杂,涉及的知识比较多,因此面向对象数据库尚未达到关系数据库的普及程度。面向对象数据模型及面向对象数据库的详细内容本书将在第 10 章介绍。

1.5 本章小结

本章概述了数据库系统的基本概念和基本结构。

本章首先介绍了计算机数据管理的发展,阐明了数据与信息的异同。数据管理是数据

处理的中心问题。随着计算机硬件和软件的发展，数据管理经历了人工管理、文件系统和数据库系统三个发展阶段。

本章介绍了数据库管理系统和数据库系统的有关概念。数据库管理系统是位于用户和操作系统之间的一层数据管理软件。数据库系统三级模式和两层映像的系统结构保证了数据库系统中能够具有较高的逻辑独立性和物理独立性。数据库语言包括数据描述语言、数据操纵语言、数据控制语言和宿主语言。

数据模型是数据库系统的核心和基础。本章介绍了组成数据模型的三个要素、概念模型和三种主要的数据库模型。这几种最常用的数据模型是层次模型、网状模型和关系模型。其中层次模型和网状模型统称为非关系模型。本章除了介绍层次模型、网状模型和关系模型的基本特点外，还简单介绍了面向对象数据模型。

1.6　习题

1.6.1　名词解释

数据、数据库、数据库管理系统、数据库系统、层次模型、网状模型、关系模型、实体-联系图、模式、内模式、外模式、DDL、DML

1.6.2　简答题

1. 什么是数据？数据和信息有什么关系？
2. 简述文件系统和数据库系统的数据管理有哪些不同。
3. 简述数据库系统的特点。
4. 数据库管理系统的主要功能有哪些？
5. 简述数据库系统的组成。
6. 简述数据库系统三级模式结构。
7. 简述数据库系统中的二级映像技术及其作用。
8. 数据库系统是如何实现数据的独立性的？
9. 层次模型和网状模型的主要特点是什么？
10. DBA 的具体职责是什么？

1.6.3　用 E-R 图表示概念模型

1. 学校中有若干系，每个系有若干班级和教研室，每个教研室有若干教师，其中有的教授和副教授每人各带若干研究生；每个班有若干学生，每个学生选修若干课程，每门课程由若干学生选修。

2. 某工厂生产若干产品，每种产品由不同的零件组成，有的零件可用在不同的产品上。这些零件由不同的原材料制成，不同零件所用的原材料可以相同。这些零件按所属的不同产品分别放在不同的仓库中，原材料按照类别放在若干仓库中。

3. 某百货商店要设计一个数据库系统，通过调研和分析知道：该商店的一个重要方面

是同供应商打交道，商店出售的商品由他们提供，各供应商供应多种商品，每种商品可能从多个供应商处购买，各供应商提供的商品价格不同；该商店有若干个部门，每个部门有一个经理和若干个雇员，每个雇员只属于一个部门；每个部门负责销售某些商品，每种商品规定只由一个部门销售；商店的顾客开订单买商品，由商店送货上门；每个顾客开的定单数不限，一个订单由顾客要求购买的若干商品和购买的数量组成。当然，同一商品可被许多顾客包括在许多订单中。

第2章 关系数据库系统

1970 年 6 月,美国 IBM 公司的 E. F. Codd 在美国计算机学会会刊(Communications of ACM)上发表题为 *A Relational Mode for Large Shared Data Banks*(大型共享系统的关系数据库的关系模型)的论文,系统而严格地提出关系模型这一概念。这篇文章首次明确而清晰地为数据库系统提出了一种崭新的模型,即关系模型,开创了数据库系统的新纪元。ACM(Association Computing Machinery)后来在 1983 年把这篇论文列为自 1958 年以来的四分之一个世纪里具有里程碑式意义的最重要的 25 篇研究论文之一。由于关系模型具有坚实的数学基础且简单明了,一经提出,立即引起学术界和产业界的广泛重视和响应,在理论与实践两个方面都对数据库技术产生了强烈的冲击。自此,基于层次模型和网状模型的数据库产品很快走向衰败,一大批基于关系模型的关系数据库系统很快被开发出来并迅速商品化,占领了市场。E. F. Codd 从 1970 年起连续发表了多篇论文,奠定了关系数据库的理论基础。DB2、Oracle、Sybase、SQL Server 等都是关系 DBMS。显然,关系数据库是采用数学方法来处理数据库中的数据的。

关系数据库系统是支持关系模型的数据库系统,其关系模型由关系数据结构、关系操作集合和完整性约束三部分组成。数据操作的方式可以用关系代数和关系演算来表示。

2.1 关系数据结构

2.1.1 关系及相关概念

在关系模型中,无论是实体还是实体之间的联系均由关系(二维表)来表示。下面从集合论的一些概念出发,给出关系数据结构的形式化定义。

1. 域

域是一组具有相同数据类型的值的集合。例如,实数、整数、正整数、小于 100 的正整数,以及英文字母集合等都可以是域。

2. 笛卡儿积

给定一组域 $D_1, D_2, \cdots, D_n$,这些域可以完全不同,也可以部分或完全相同。$D_1, D_2, \cdots, D_n$ 的笛卡儿积为:

$$D_1 \times D_2 \times \cdots \times D_n = \{(d_1, d_2, \cdots, d_n) \mid d_i \in D_i, i = 1, 2, \cdots, n\},$$

其中每一个元素$(d_1,d_2,\cdots,d_n)$称做一个n元组,或简称为元组。元素中的每一个值d_i为一个分量。

若$D_i(i=1,2,\cdots,n)$为有限集,其基数为$m_i(i=1,2,\cdots,n)$,则$D_1\times D_2\times\cdots\times D_n$的基数为$m=\prod_{i=1}^{n} m_i$。

笛卡儿积可以表示为一个二维表,表中每行为一个元组,每列对应一个域。

例如,有三个域:D_1=书名集合 BOOKNAME={C 语言程序设计、数据结构、数据库技术},D_2=作者集合 AUTHOR={王一,李二},D_3=出版社集合 PUBLISHER={清华大学出版社,天津大学出版社,科学出版社}。

$D_1\times D_2\times D_3$的笛卡儿积如表 2-1 所示,共有 18(3×2×3)个元组。

表 2-1　$D_1\times D_2\times D_3$ 的笛卡儿积

BOOKNAME	AUTHOR	PUBLISHER
C 语言程序设计	王一	清华大学出版社
C 语言程序设计	王一	天津大学出版社
C 语言程序设计	王一	科学出版社
C 语言程序设计	李二	清华大学出版社
C 语言程序设计	李二	天津大学出版社
C 语言程序设计	李二	科学出版社
数据结构	王一	清华大学出版社
数据结构	王一	天津大学出版社
数据结构	王一	科学出版社
数据结构	李二	清华大学出版社
数据结构	李二	天津大学出版社
数据结构	李二	科学出版社
数据库技术	王一	清华大学出版社
数据库技术	王一	天津大学出版社
数据库技术	王一	科学出版社
数据库技术	李二	清华大学出版社
数据库技术	李二	天津大学出版社
数据库技术	李二	科学出版社

表 2-1 所示的笛卡儿积表示了书名、作者、出版社所有可能的组合。

3. 关系

$D_1\times D_2\times\cdots\times D_n$的子集称做在域$D_1,D_2,\cdots,D_n$上的关系,用$R(D_1,D_2,\cdots,D_n)$表示。这里$R$为关系名,$n$是关系的目或度。

关系是笛卡儿积的子集,因此关系也是一个二维表。在上例的笛卡儿积中取出某些元组可以构成关系 BAP,如表 2-2 所示。

关系 BAP 的元组是从笛卡儿积中抽取的,但不是任意抽取的,例如元组(C 语言程序设计,王一,科学出版社)就不能成为关系 BAP 的元组,因为它与元组(C 语言程序设计,王一,

清华大学出版社)相矛盾。

表 2-2 关系 BAP

BOOKNAME	AUTHOR	PUBLISHER
C语言程序设计	王一	清华大学出版社
数据结构	王一	天津大学出版社
C语言程序设计	李二	科学出版社
数据库技术	王一	科学出版社

4. 关键字

关系的每一列称为一个属性。若关系中的某一属性组能唯一地标识一个元组,则该属性称为候选码(Candidate Key)。例如在某个图书馆里,馆藏的图书都有唯一的编码,这个编码就是候选码。

若一个关系有若干个候选码,则选定其中一个为主码(Primary Key),例如,馆藏的图书除了在馆内唯一的编码外,还有一个唯一的ISBN,因此ISBN也是候选码,但是,可以制定图书编号为主属性。

有的时候主码是由多个属性组成的。主码的诸属性称为主属性,不包含在任何候选码中的属性为非码属性。例如在借阅(图书证号,图书编号,借阅日期,应还日期)这个关系中,图书证号+图书编号为主码,所以图书证号和图书编号均为主属性,而借阅日期和应还日期为非码属性。

5. 关系的约束

从数学上看,关系就是一个元数为$K(K\geqslant 1)$的元组的集合。从应用的角度看,关系是一种规范化的表格,满足以下约束:

(1) 列是同质的,即列中的每一个分量来自同一个域。

(2) 不同的列可以来自相同的域,但属性名不能相同,即同一个关系中,属性名不能相同。

(3) 列的顺序无关紧要,可以任意交换。

(4) 行的顺序无关紧要,可以任意交换。

(5) 关系中没有重复元组,即任意两个元组都不能完全相同。

(6) 分量必须取原子值,即每一个分量都是不可分的数据项。

其中:约束(5)保证了元组引用的确定性和唯一性;约束(1)(2)保证了列的引用的确定性和完整性;约束(6)保证了行和列同时引用的属性值的唯一性;约束(3)(4)表明构造关系时的自由性,出现新的属性或元组时,可随时增加在后面,无须调整顺序。

6. 关系的三种类型

(1) 基本关系

基本关系(通常又称基本关系或基表)是实际存在的表,它是实际存储数据的逻辑表示,相当于模式。

(2) 视图表

视图表是由基本表或其他视图表导出的表。由于视图表不是实际的表，它本身不独立存储在数据库中，数据库中只存放视图表的定义，因此视图表是虚表。

(3) 查询表

查询表是查询结果对应的表，在数据库中仅存放查询表的定义，查询运行后才能生成查询表。

2.1.2 关系模式

前面已经指出了关系是一个二维表，这种二维表可以存放两类信息，即实体本身的信息和实体间的联系。这种由表格数据来表示和实现实体间关系的做法是符合自然规律的，是关系模式的本质所在。

1. 关系模式的定义

关系模式是对关系的描述和抽象，关系是关系模式在某一时刻的状态或内容。换言之，关系模式是型，关系是值。关系模式是静态的、稳定的，关系是动态的，不同时刻关系模式中的关系可能会不同。关系模式和关系往往统称为关系。

从形式化的角度可以将关系模式定义成一个五元组：$R(U, D, \mathrm{DOM}, F)$。其中各元组表达意义如下：

R——关系名；

U——属性组，即组成 R 的全部属性的集合；

D——域的集合，即属性取值范围的集合；

DOM——U 与 D 之间的映像；

F——属性组 U 上的数据依赖关系的集合。

关系模式可以简记为 $R(U)$ 或 $R(A_1, A_2, \cdots, A_n)$，其中 R 为关系名，$A_1, A_2, \cdots, A_n$ 为属性名。

2. 关系模式的优点

(1) 数据结构简单

关系数据模型的本质就是二维表，其中公共属性名指示着各表格间的联系。

(2) 可以直接处理多对多的关系

层次和网状模型不能直接处理多对多的关系，而关系模型由于采用表格，可以直接表示两实体间的联系，所以能直接处理多对多的关系。

(3) 能够一次提供一个元组集合

每一个操作命令都可以得到满足某种条件的所有记录，而层次与网状模型每一次操作只能得到一条记录，如果要得到所有满足条件的记录，则需要借助主语言并配合 DML 的命令。

(4) 数据独立性较高

用户只需指出他们所要存放的数据类型、数据长度等数据本身的特征，而不必涉及这些数据的物理存放，因而数据独立性较高；而在层次和网状模型中，用户或多或少地都要对其

数据的物理组织进行干预。

(5) 坚实的理论基础

关系模型建立在集合代数理论的基础上，并且近几年又投入了大量的人力物力，使得关系理论趋于完善。而层次和网状模型的系统研制和数据库设计都无一定的理论指导，仅凭设计者的经验和技术水平。因此，系统研制和应用设计都较为盲目。由于关系模型的系统研制和应用设计有了理论指导，所以在层次和网状模型的系统中出现的很多问题在关系模型中都可以避免。

3. 关系模式的缺点

(1) 查询效率较低

关系模型的 DBMS 能提供较高的数据独立性以及非过程化的查询语言，因此，系统的负担就很重，过去需要由程序员做的工作，现在全部由系统包办代替，其中会影响效率的操作是笛卡儿积运算和两个表的连接。

(2) 关系 DBMS 实现较困难

为提高效率，必须使查询优化，而这一工作是复杂的。

(3) 关系 DBMS 实现要求规范化

关系模型的 DBMS 要求程序员和 DBA 应熟悉关系数据库设计理论，能够熟练地进行关系模式规范化工作，以便充分发挥关系 DBA 的功能。

4. 关系系统的六大目标

关系理论的创造人 E. F. Codd 提出的关系系统的六大目标已大部分实现，这六大目标是：

(1) 提供高度的数据独立性；

(2) 提供严格而简明的数据视图；

(3) 简化 DBA 的工作；

(4) 建立理论基础；

(5) 把事务管理和文件管理结合起来；

(6) 把基于数据的应用程序设计提高到一个新的水平，即操作对象是记录集合，而不是单个记录。

2.1.3 关系数据库

对应于一个关系模式的所有关系的组合称为一个关系数据库。关系数据库也有型和值之分。关系数据库的型是对关系数据库的描述，包括若干域和定义在这些域上的关系模式，也可称为关系数据库模式。关系数据库的值，是某一时刻对应的关系的集合，也可称为关系数据库。通常，关系数据库模式和关系数据库统称为关系数据库。

关系数据库要求让用户所感觉到的数据库是一张张表的集合。在关系数据库中，表是逻辑结构而不是物理结构。实际上，关系数据库系统在物理层可以使用任何有效的存储结构来存储数据，如有序文件、索引、哈希表、指针等。因此，表是对物理存储数据的一种抽象表示，是对很多存储细节的抽象，如存储记录的位置、记录的顺序、数据值的表示等，以及记

录的访问结构，如索引等。它们对用户来说都是不可见的。

关系模式与关系数据库中术语的对应关系如下：

关系↔表

元组↔记录

属性↔字段

关系模式↔数据库

2.2 关系操作集合

2.2.1 基本关系操作

关系数据操作的对象都是关系，其操作结果仍为关系，即集合式操作。既然关系模型是基于坚实的数学基础的，那么关系操作与数学就有着紧密的联系，因此关系数据库中常用的关系数据操作有：

（1）数据查询：主要操作涉及关系属性的指定、元组的选择、两个关系的合并与连接等，可通过选择、投影、连接、除、并、交、差等来定义和导出。

（2）数据更新：主要操作涉及在关系中插入、删除元组及修改元组的内容等。

而表达或者称为描述关系操作的关系数据语言可以分为三类，如表 2-3 所示。

表 2-3 关系数据语言分类

关系数据语言	1. 关系代数语言		例如：ISBL
	2. 关系演算语言	元组关系演算语言	例如：ALPHA，QUEL
		域关系演算语言	例如：QBE
	3. 具有关系代数和关系演算双重特点的语言		例如：SQL

2.2.2 关系数据语言分类

关系模型与其他数据模型相比，最具特色的是它的数据操纵语言。该语言灵活方便，表达能力和功能都很强。它是非过程化的。用户只要指出做什么，不必指出怎样做，它就可以独立完成，它提供给用户的是一个记录集，而不是一个记录。

关系模型的数据操纵语言之所以有这些特点，主要原因是：采取了最简单、最规范的数据结构，并运用了先进的教学工具——集合运算和谓词演算，又创造了几个特殊运算——投影、选择和连接，使用这些运算可以对二维表进行任意分割和组装，随时构造出各式各样的用户所需要的表格，即关系。

作为用户，没必要知道系统在内部是怎样分割和组装的，只需要指出所要用到的数据和限制条件。当然，作为系统分析员和系统设计者，需要了解内部的情况。

1. 关系代数

关系代数用代数方式对关系进行运算来表达查询要求，它是以集合操作为基础的运算，即操作的对象结果都是集合。查询表达式中需要指明操作的先后顺序。

2. 关系演算

关系演算用谓词方式来表达查询要求。关系演算又可按谓词变元的基本对象是元组变量还是域变量分为元组关系演算和域关系演算两种。关系代数、元组关系演算和域关系演算这三种语言在表达能力上是完全等价的。

关系代数、元组关系演算和域关系演算均是抽象的查询语言,这些抽象的语言与具体的DBMS中实现的实际语言并不完全一样,但它们能用做评估实际系统中查询语言能力的标准或基础。

3. SQL

介于关系代数和关系演算之间的语言SQL(Structured Query Language,结构化查询语言),是由IBM公司在研制System R时提出的,SQL不仅具有丰富的查询功能,而且具有数据定义和数据控制功能,是集数据查询语言(DQL)、数据定义语言(DDL)、数据操纵语言(DML)和数据控制语言(DCL)于一体的关系数据语言。SQL充分体现了关系数据语言的特点和优点,是关系数据库的标准语言。

在实际应用的关系数据库管理系统中所使用的查询语言功能十分强大,除了提供关系代数与关系演算的功能外,还包含了许多附加功能,如聚集函数、关系赋值、算术运算等,甚至还具备一定的编程能力。目前使用最广泛的SQL语言不但同时具有关系代数语言和关系演算语言的双重特点,而且具有数据定义和数据控制功能,是集查询、DDL、DML和DCL于一体的关系数据语言。

2.3 完整性约束

关系模型的完整性规则是对关系的某种约束条件。关系模型中可以有三类完整性约束:实体完整性、参照完整性和用户定义的完整性。其中实体完整性和参照完整性是关系模型必须满足的完整性约束条件,称为关系的两个不变性,应该由关系系统自动支持。下面分别介绍这三类完整性约束。

2.3.1 实体完整性

所谓实体完整性,就是指一个关系模型中的所有元组都是唯一的,没有两个完全相同的元组,也就是说一个二维表中没有两个完全相同的行,也称为行完整性。在数据输入或修改过程中,完全相同的行不能存储。一般来说,元组对应现实世界的一个实体,所以称这种约束为实体完整性约束。在具体实现过程中,用户可以为自己的关系模型定义一个唯一性关键字,一般的关系数据库系统都会阻止两个关键字相同的元组存入系统中。

实体完整性规则:若属性(组)A是基本关系R主码上的属性,则属性A不能取空值。

说明:

(1) 实体完整性规则是对基本关系的约束和限定。一个基本关系表通常对应一个实体集。例如,读者关系对应读者集合。

(2) 现实世界中的实体是可以区分的,它们具有一种唯一性质的标识。例如,读者的图书证编号、图书的图书编号等。

(3) 在关系模型中,主码作为唯一的标识且不能为空。组成主码的每一个属性都不能取空值。

注意:有多个候选码时,主码以外的候选码上可取空值。

在表 2-4 所示的“读者”关系中,图书证编号是主码(因为图书证编号唯一确定这个读者的属性),也是主属性,但是第四条记录的讲师编号是空值,因此关系不满足实体完整性规则。当然,目前的关系 DBMS 都会自动帮助用户完成完整性的约束检查,即自动支持满足实体完整性规则。

表 2-4 读者表

图书证编号	姓名	类型	性别	出生日期
S001	王红	大本	女	1989-02-13
S002	张静	研究生	女	1984-06-20
T001	舒欣	副教授	女	1972-03-26
	李雷	讲师	男	1975-09-16

2.3.2 参照完整性

现实世界中的实体集之间往往存在某种联系,在关系模型中,实体集及实体集间的联系都是用关系来描述的。这样就自然存在着关系与关系间的引用。

例如,有如下学生和院系两个实体集合。

学生(学号,姓名,性别,年龄,院系编号),其中学号为主码。

院系(院系编号,系主任,办公地点),其中院系编号为主码。

这两个关系之间存在着属性的引用,即学生关系引用了院系关系的主码“院系编号”。

显然,学生关系中“院系编号”的值必须是在院系表中确实存在的,即学院关系中有该院系的记录。这说明学生关系中某个属性的取值要参照院系关系的属性取值,即本学校的学生就读的不可能是本学校没有开设的院系。在这种情况下,“院系编号”不是学生关系的主关键字,但是它是院系关系的主码,故称“院系编号”是学生关系的外码。

又如有以下实体集:

学生(学号,姓名,性别,年龄,院系编号),其中学号为主码。

课程(课程号,名称,学分,学时),其中课程号为主码。

选课(学号,课程号,成绩),其中学号与课程号联合为主码。

引用关系是指关系中某属性的值需要参照另一关系的属性来取值。设选课关系引用了学生关系的主码“学号”和课程关系的主码“课程号”。显然,选课关系中的“学号”值必须是学生关系中实际存在的某学号;选课关系中的“课程号”值也必须是已开设的某课程号。换言之,选课关系中某些属性的值需要参照学生关系及课程关系对应的属性内容来取值。称选课关系为参照关系(或依赖表),学生关系和课程关系为目标关系(或被参照关系)。

不仅两个或两个以上的关系之间可以存在引用关系,而且同一关系内部属性间也可能存在引用关系。课程关系中的“前导课编号”必须参照课程关系中的“课程号”,因此,课程关

系既是参照关系，又是目标关系。

参照完整性的定义：设有基本关系 R、S（可为同一关系）。若 F 是 R 的一个属性（组），但不是 R 的码，如果 F 与 S 的主码 K 相对应，则称 F 是 R 的外码，并称 R 为参照关系，S 为目标关系。

不难看出，主码 K 和外码 F 必须定义在相同域上，它们相对应即有引用关系。

下面的参照完整性规则界定了主码与外码间的引用关系。

参照完整性规则：若属性（或属性组）F 是基本关系 R 的外码，它与基本关系 S 的主码 K_s 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须为：

- 或者取空值（F 的每个属性值均为空值）。
- 或者等于 S 中某个元组的主码值。

简单说来，即当一个数据表中有外部关键字（即该列是另外一个表的关键字）时，外部关键字列的所有值，都必须出现在其所对应的表中，这就是参照完整性的含义。其作用一般有如下 3 个方面：

(1) 禁止从表中插入主表中不存在的关键字的数据行。

(2) 禁止会导致从表中的相应值孤立的主表中外部关键字值的改变。

(3) 禁止删除与从表中有对应记录的主表记录。

参照完整性约束保证了两个有关联的表相互连接的正确性。

参照完整性又称引用完整性，它定义了外码与主码之间的引用规则。

外码与主码提供了一种表示元组之间联系的手段。外码要么空缺，要么引用一个实际存在的主码值。

在上面的例子中，学生关系中每个元组的“院系号”只能取下面两类值：

- 空值，表示该学生刚被录取，尚未分配到院系。
- 非空值，这时该值必须是院系关系中某个院系号的值。

同理，课程关系中每个元组的“前导课编号”只能取某一课程号值或为空值，若为空值则表示该没有前导课等。

在选课关系的例子中，选课关系中的“学号”与学生关系的主码“学号”相对应，因此是外码；选修关系中的“课程号”与课程关系的主码“课程号”相对应，故也是外码。根据参照完整性规则，它们要么空缺，要么引用对应关系中实际存在的主码值。但由于选修关系自身的主码为“学号”与“课程号”，又根据实体完整性规则可知，它们均不能为空。故只能取对应目标关系中的实际值，而不能取空值。

2.3.3 用户定义完整性

除实体完整性和参照完整性约束之外，关系数据库系统还应该能够根据其应用环境的不同，让用户添加一些特殊的约束条件来满足其需要，由此引出用户定义完整性的概念。所谓用户定义完整性就是针对某一具体应用的关系数据库的具体的应用需求，而对其数据进行的必要的约束条件，它反映某具体应用所涉及的数据必须满足的要求。例如，学生关系中的年龄在 15～75 之间，选修关系中的成绩在 0～100 之间；更新职工表时，工资、工龄等属性值通常只增加不减少；等等。这类约束条件不必放在应用程序中去检验，而由数据库系

统去检验这一完整性约束，以减少应用程序的工作量。

关系模型应提供定义和验证这类完整性的机制，以便用统一的系统方法处理它们，而不要让应用程序承担这一功能。关系数据库系统一般包括以下几种用户定义的完整性约束：

(1) 定义属性是否为空值；

(2) 定义属性值的唯一性；

(3) 定义属性的取值范围；

(4) 定义属性的默认值；

(5) 定义属性间函数依赖关系。

2.4 关系代数

2.2 节讲述了描述关系操作的关系数据语言可以分为三类：关系代数、关系演算和介于关系代数与关系演算之间的语言 SQL。下面专门讲述用对关系进行运算来表达查询要求的关系代数。关系代数的运算对象是关系，运算结果也是关系。关系代数用到的运算符包括四类：集合运算符、专门的关系运算符、算术比较运算符和逻辑运算符，其中比较运算符和逻辑运算符是用来辅助运算的专门关系运算符，如表 2-5 所示。

表 2-5 关系运算符

传统集合运算符	专门的关系运算符	
$\cup$：并	σ：选择	辅助专门关系运算符
$\cap$：交	$\prod$：投影	比较运算符：$>$、$\geqslant$、$<$、$\leqslant$、$=$、$\neq$
$-$：差	$\bowtie$：连接	逻辑运算符：$\neg$、$\wedge$、$\vee$
$\times$：广义笛卡儿积	$\div$：除	

传统的集合运算包括并、交、差和广义笛卡儿积等。集合运算把关系看做元组的集合，从水平(行)方向进行运算，广义笛卡儿积把两个关系的元组以所有可能的方式组成对。

专门的关系运算包括选择、投影、连接和除等，既从行又从列的方向进行运算。“选择”会删除某些行；“投影”会删除某些列；各种连接运算将两个关系的元组中有选择地组成对，构成一个新关系。

2.4.1 传统的集合运算

传统的集合运算包括并、交、差、广义笛卡儿积四种运算。其中，并、交、差要求 R 和 S 是相容的。

设关系 R 和 S 是相容的，是指关系 R 和关系 S 具有相同的目 n(即两个关系都有 n 个属性)，且相应的属性取自同一个域，其结果关系仍为 n 目关系。

1. 并

R 和 S 的并(UNION)记作：$R \cup S = \{ t | t \in R \vee t \in S \}$

$R \cup S$ 的结果生成一个新关系，由属于 R 的或属于 S 的所有元组组成，如图 2-1(a)所示。$R \cup S$ 的具体算例如图 2-2 所示。

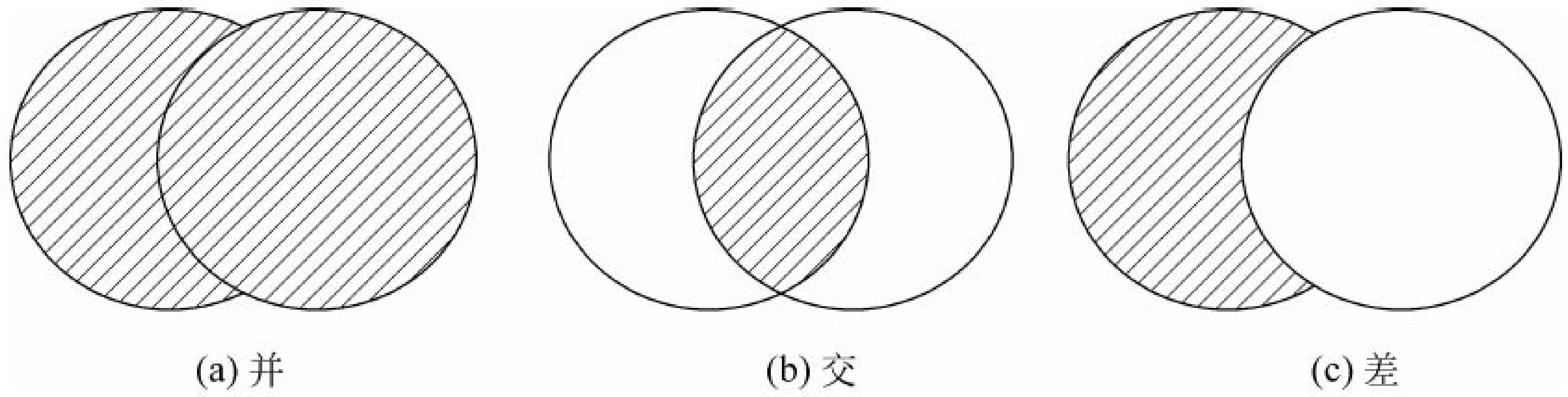

图 2-1 传统集合运算并、交、差的图示

注意：

(1) R 和 S 的并，$R \cup S$，是在 R 或 S 或两者中的元组的集合；

(2) 一个元组在并集中只出现一次；

(3) R 和 S 必须同类型(属性集相同、次序相同，但属性名可以不同)。

2. 交

R 和 S 的交(INTERSECT)记作：$R \cap S = \{ t | t \in R \wedge t \in S \}$

$R \cap S$ 的结果生成一个新关系，由属于 R 的或属于 S 的所有元组组成，如图 2-1(b)所示。$R \cap S$ 的具体算例如图 2-2 所示。

注意：

(1) R 和 S 的交，$R \cap S$，是在 R 和 S 中都存在的元组的集合；

(2) 一个元组在交集中只出现一次；

(3) R 和 S 必须同类型(属性集相同、次序相同，但属性名可以不同)。

3. 差

R 和 S 的差(MINUS)记作：$R - S = \{ t \mid t \in R \wedge t \notin S \}$

$R - S$ 的结果生成一个新关系，由属于 R 的或属于 S 的所有元组组成，如图 2-1(c)所示。$R - S$ 的具体算例如图 2-2 所示。

注意：

(1) R 和 S 的差，$R - S$，是在 R 中而不在 S 中的元组的集合；

(2) R 和 S 必须同类型(属性集相同、次序相同，但属性名可以不同)。

两关系的交集可以通过差运算导出：$R \cap S = R - (R - S)$

4. 广义笛卡儿积

两个分别为 n 目和 m 目的关系 R 和 S 的广义笛卡儿积是一个($n \times m$)列的元组的集合。元组的前 n 列是关系 R 的一个元组，后 m 列是关系 S 的一个元组。若 R 有 $k1$ 个元组，S 有 $k2$ 个元组，则关系 R 和关系的广义笛卡儿积有 $k1 \times k2$ 个元组，记为 $R \times S$，可表示为 $R \times S = \{ t_r t_s | t_r \in R \wedge t_s \in S \}$

即结果是 R 中的每一个元组分别与 S 中的所有元组连接构成新关系 $R \times S$ 的元组。

R

A	B	C
$a1$	$b1$	$c1$
$a1$	$b2$	$c1$
$a1$	$b3$	$c2$
$a2$	$b1$	$c2$
$a2$	$b2$	$c2$

S

A	B	C
$a1$	$b2$	$c1$
$a2$	$b1$	$c2$
$a2$	$b3$	$c2$

R∪S

A	B	C
$a1$	$b1$	$c1$
$a1$	$b2$	$c1$
$a1$	$b3$	$c2$
$a2$	$b1$	$c2$
$a2$	$b2$	$c2$
$a2$	$b3$	$c2$

R∩S

A	B	C
$a1$	$b2$	$c1$
$a2$	$b1$	$c2$

R−S

A	B	C
$a1$	$b1$	$c1$
$a1$	$b3$	$c2$
$a2$	$b2$	$c2$

R×S

A	B	C	A	B	C
$a1$	$b1$	$c1$	$a1$	$b2$	$c1$
$a1$	$b1$	$c1$	$a2$	$b1$	$c2$
$a1$	$b1$	$c1$	$a2$	$b3$	$c2$
$a1$	$b2$	$c1$	$a1$	$b2$	$c1$
$a1$	$b2$	$c1$	$a2$	$b1$	$c2$
$a1$	$b2$	$c1$	$a2$	$b3$	$c2$
$a1$	$b3$	$c2$	$a1$	$b2$	$c1$
$a1$	$b3$	$c2$	$a2$	$b1$	$c2$
$a1$	$b3$	$c2$	$a2$	$b3$	$c2$
$a2$	$b1$	$c2$	$a1$	$b2$	$c1$
$a2$	$b1$	$c2$	$a2$	$b1$	$c2$
$a2$	$b1$	$c2$	$a2$	$b3$	$c2$
$a2$	$b2$	$c2$	$a1$	$b2$	$c1$
$a2$	$b2$	$c2$	$a2$	$b1$	$c2$
$a2$	$b2$	$c2$	$a2$	$b3$	$c2$

图 2-2 关系 R 和 S 的传统集合运算算例

2.4.2 专门的关系运算

专门的关系运算包括选择、投影、连接、除等。

在学习专门的关系运算之前，先了解一些术语及其表示。

- 设关系模式为 $R(A_1, A_2, \cdots, A_n)$，它的一个关系设为 t，$t \in R$ 表示 t 是 R 的一个元组，$t[A_i]$则表示元组 t 中相对于属性 A_i 的一个分量。
- 若 $A=\{A_{i1}, A_{i2}, \cdots, A_{ik}\}$其中 $A_{i1}, A_{i2}, \cdots, A_{ik}$ 是 $A_1, A_2, \cdots, A_n$ 中的一部分，则称为属性列或属性组。$t[A]=(t[A_{i1}], t[A_{i2}], \cdots, t[A_{ik}])$ 表示元组 t 在属性列 A 上诸

分量的集合。$\overline{A}$ 则表示 $\{A_1, A_2, \cdots, A_n\}$ 中去掉 $\{A_{i1}, A_{i2}, \cdots, A_{ik}\}$ 后剩余属性组。

- R 为 n 目关系，S 为 m 目关系，$t_r \in R$，$t_s \in S$，$\widehat{t_r t_s}$ 称为元组的连接（Concatenation）它是一个 $n+m$ 列的元组，前 n 个分量为 R 中的一个 n 元组，后 m 个分量为 S 的一个 m 元组。
- 给定一个关系 $R(X,Z)$，X 和 Z 为属性组。当 $t[X]=x$ 时，x 在 R 中的象集定义为：

$$Z_x = \{t[Z] \mid t \in R, t[X] = x\}$$

它表示 R 中属性组 X 上值为 x 的诸元组在 Z 上分量的集合。

对于表 2-6 所示的关系，$X=\{S\}$，则：

001 在 R 中的象集 $Z_{001}=\{(C1,80),(C2,85),(C3,90)\}$

002 在 R 中的象集 $Z_{002}=\{(C2,95)\}$

003 在 R 中的象集 $Z_{003}=\{(C2,90),(C3,85)\}$

表 2-6 关系 *SCG*

S	*C*	*G*	*S*	*C*	*G*
001	*C*1	80	002	*C*2	95
001	*C*2	85	003	*C*2	90
001	*C*3	90	003	*C*3	85

1. 选择

选择是对关系的水平分解运算，是对关系进行的操作的元组组成新关系。选择运算表示为：

$$\sigma_F(R) = \{t \mid t \in R \land F(t) = T\}$$

其中：σ 为选择运算符；R 是关系名；F 为逻辑表达式。F 中的运算对象通常为常量、函数或元组分量，运算符为比较运算符和逻辑运算符，表示的条件取值为 T 或 F。

选择运算是从关系的水平方向进行的运算，是从关系 R 中选择满足给定条件的所有元组。选择运算如图 2-3(a)所示。

2. 投影

投影是对关系的垂直分解运算，即从关系的属性集中选择属性子集，构成的元组组成一个新关系。投影操作表示为：

$$\pi_A(R) = \{t[A] \mid t \in R\}$$

其中：π 为投影运算符；R 是关系名；A 表示关系 R 中的属性子集合。该操作从关系 R 中移出部分列，只保留 A 列组成一个新的关系，并去掉重复的元组。新关系中的属性值来自原关系中相应的属性值，列的次序在新关系中可以重新排列。

投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组，因为取消了某些属性列后，就可能出现重复行，应取消这些完全相同的行。投影运算如图 2-3(b)所示。

可以同时对某个关系进行选择和投影操作，得出想要的结果，即进行所谓的查询操作。

注意：当要对某一关系进行选择和投影操作时，一定要先选择后投影，因为投影是对列的筛选，如果先投影，就会把要进行选择的条件属性切除掉，使得选择操作无法进行。

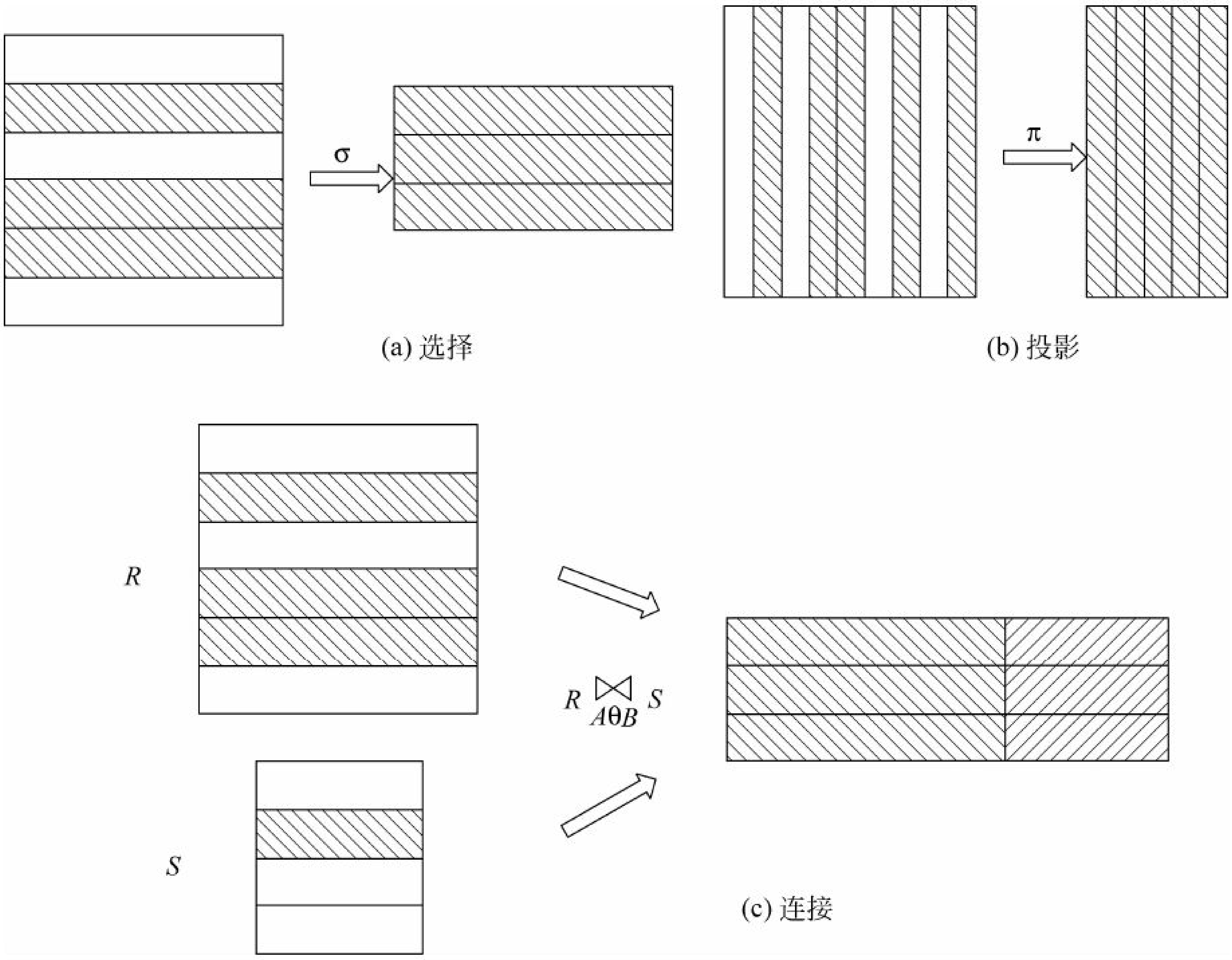

图 2-3　选择、投影、连接的图示

3. 连接

连接是从两个关系的广义笛卡儿积中选择属性间满足一定条件的元组的运算。表示为：

$$R \underset{A\theta B}{\bowtie} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B]\}$$

其中，A 和 B 分别为 R 和 S 上可比的属性组；θ 是比较运算符。连接运算从 R 和 S 的笛卡儿积 $R \times S$ 中选取在 A 属性组（R 关系）上的值与在 B 属性组（S 关系）上的值满足比较关系 θ 的元组，如图 2-4 所示。

连接运算中有两种最为重要的也是很常用的连接，一种是等值连接，一种是自然连接。

（1）等值连接与自然连接

θ 为"＝"的连接运算，称为等值连接。它是从关系 R 与 S 的广义笛卡儿积中选取 A、B 属性值相等的那些元组，如图 2-4 所示。等值连接为 $R \underset{A=B}{\bowtie} S$，记作：

$$R \underset{A=B}{\bowtie} S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B]\}$$

自然连接是一种特殊的等值连接，它要求等值连接的连接属性是相同属性（或属性组），即 R 和 S 具有相同的属性组，且该组属性具有相同的列值，则该等值连接称为自然连接 $R \bowtie S$，记为：

$$R \bowtie S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[B] = t_s[B]\}$$

自然连接的结果关系中要去掉重复的属性。假定 R 和 S 具有的相同属性 A，且 R 有 n 个属性，S 有 m 个属性，自然连接的结果有$(n+m-1)$个属性。

R

A	B	C
$a1$	2	$c1$
$a1$	5	$c1$
$a1$	9	$c2$
$a2$	9	$c2$
$a2$	12	$c2$
$a3$	2	$c4$

S

C	D
$c1$	5
$c2$	7
$c3$	9

$R \underset{B>D}{\bowtie} S$

A	B	R.C	S.C	D
$a1$	9	$c2$	$c1$	5
$a2$	9	$c2$	$c1$	5
$a2$	12	$c2$	$c1$	5
$a1$	9	$c2$	$c2$	7
$a2$	9	$c2$	$c2$	7
$a2$	12	$c2$	$c2$	7
$a2$	12	$c2$	$c3$	9

$R \underset{R.C=S.C}{\bowtie} S$

A	B	R.C	S.C	D
$a1$	2	$c1$	$c1$	5
$a1$	5	$c1$	$c1$	5
$a1$	9	$c2$	$c2$	7
$a2$	9	$c2$	$c2$	7
$a2$	12	$c2$	$c2$	7

$R \bowtie S$

A	B	C	D
$a1$	2	$c1$	5
$a1$	5	$c1$	5
$a1$	9	$c2$	7
$a2$	9	$c2$	7
$a2$	12	$c2$	7

图 2-4 关系 R 和 S 的连接运算

等值连接、自然连接的示例如图 2-4 所示，其中关系 R 和关系 S 有共同属性 C，假设 C 是同值的，即两个关系的 C 是来自同一个域并且意义相同，自然连接的结果比等值连接少一个或多个属性。

本例中采用的连接方法是从理论上分析的方法，由于两个关系取广义笛卡儿积运算非常费时，实际的 DBMS 所采用的连接算法更有可能是优化的算法。

注意：一般对表的查询是结合选择投影和自然连接进行的。可以同时对某个关系进行连接、选择和投影操作，得出想要的结果，即进行所谓的查询操作。

(2) 外连接

外连接运算是对连接运算的扩展，可以处理缺失信息。自然连接是选择两个关系在公共属性组上值相等的元组构成的，因此与公共属性组值不同的 R 与 S 的元组被忽略了。例如图 2-4 所示的自然连接运算，R 中的元组($a3$,2,$c4$)和 S 中的元组($c3$,9)就被忽略掉了。如果把舍弃的元组也保存在结果关系中，而在其他属性上填空值(Null)，那么这种连接就叫做外连接(OUTER JOIN)。

如果只把左边关系 R 中要舍弃的元组保留，就叫做左外连接(LEFT OUTER JOIN 或 LEFT JOIN)；如果只把右边关系 S 中要舍弃的元组保留，就叫做右外连接(RIGHT OUTER JOIN 或 RIGHT JOIN)。

图 2-4 中的关系 R 与关系 S 的外连接、左外连接、右外连接运算如图 2-5 所示。

4. 除

除运算是同时从关系的水平方向和垂直方向进行运算。对于给定关系 $R(X,Y)$ 和 $S(Y,Z)$，其中 X、Y、Z 为属性组，R 中的 Y 与 S 中的 Y 可以有不同的属性名，但必须出自相同的域集。$R \div S$ 应当满足元组在 X 上的分量值 x 的象集，Y_x 包含关系 S 在属性 Y 上的投

影的集合。$R \div S$ 记作：

$$R \div S = \{t_r[X] \mid t_r \in R \land \pi_Y(S) \subseteq Y_x\}$$

外连接

A	B	C	D
a1	2	c1	5
a1	5	c1	5
a1	9	c2	7
a2	9	c2	7
a2	12	c2	7
a3	2	c4	NULL
NULL	NULL	c3	9

左外连接

A	B	C	D
a1	2	c1	5
a1	5	c1	5
a1	9	c2	7
a2	9	c2	7
a2	12	c2	7
a3	2	c4	NULL

右外连接

A	B	C	D
a1	2	c1	5
a1	5	c1	5
a1	9	c2	7
a2	9	c2	7
a2	12	c2	7
NULL	NULL	c3	9

图 2-5　关系 R 和 S 的外连接运算

其中，Y_x 为 x 在 R 中的像集，$x = t_r[X]$

如图 2-6 所示，计算在关系 $R \div S$。

解：根据除法定义，此题 X 为属性 AB，Y 为属性 CD。$R \div S$ 应当满足元组在属性 AB 上的分量值的像集 Y，包含关系 S 在 CD 上投影的集合。

关系 S 在 Y 上投影的集合为 $\pi_{CD}(S) = \{(c, d), (e, f)\}$，对于关系 R，属性组 X，即 AB 可以取三个值$\{(a, b), (b, d), (c, k)\}$，它们的像集分别为：

(a, b)的像集为$\{(c, d), (e, f), (h, k)\}$

(b, d)的像集为$\{(e, f), (d, l)\}$

(c, k)的像集为$\{(c, d), (e, f)\}$

S 在(a, b)和(c, k)的像集包含了 S 在 CD 属性组上的投影$\{(c, d), (e, f)\}$

所以 $R \div S = \{(a, b), (c, k)\}$。

关系 R

A	B	C	D
a	b	c	d
a	b	e	f
a	b	h	k
b	d	e	f
b	d	d	l
c	k	c	d
c	k	e	f

关系 S

C	D
c	d
e	f

R÷S

A	B
a	b
c	k

图 2-6　关系 $R \div S$ 运算

2.4.3　综合算例

假设有如下基本关系（如图 2-7 所示）：

学生关系 S(学号 Sno、姓名 Sname、性别 Ssex、年龄 Sage、所在班 Sclass)；

课程关系 C(课程号 Cno、名称 Cname、先行课 Cpno、学分 Ccredit)；

选课关系 SC(学号 Sno、课程号 Cno、成绩 Grade)。

【例 2.1】 查询软件工程 0601 班的(SE0601 班)全体学生

$$\sigma_{Sclass='SE0601'}(S) \text{ 或 } \sigma_{5='SE0601'}(S)$$

运算结果为：

Sno	Sname	Ssex	Sage	Sclass
2006111121	王昕	F	21	SE0601
2006111122	李伟	M	20	SE0601

【例 2.2】 查询所有课程的名称和学分

$$\pi_{Cname,\ Ccredit}(C) \text{ 或 } \pi_{2,4}(C)$$

运算结果为：

Cname	Ccredit	Cname	Ccredit
高等数学	6	数据库原理	3
程序设计基础	6	面向对象程序设计	4
数据结构	6	操作系统	4
计算机原理	4	计算机网络技术	3

关系 *S*

Sno	Sname	Ssex	Sage	Sclass
2007111001	张宇	M	20	CS0701
2007111002	赵娜	F	19	CS0701
2006111121	王昕	F	21	SE0601
2006111122	李伟	M	20	SE0601
2009111001	张莉	F	18	MA0901
2009112001	李彤	F	19	MA0901
…	…	…	…	…

关系 *C*

Cno	Cname	Cpno	Ccredit
01	高等数学		6
02	程序设计基础	01	6
03	数据结构	02	6
04	计算机原理		4
05	数据库原理	03	3
06	面向对象程序设计	01	4
07	操作系统	03	4
08	计算机网络技术	07	3
…	…	…	…

关系 *SC*

Sno	Cno	Grade
2007111001	03	64
2007111001	05	81
2007111001	06	72
2006111121	02	92
2006111121	03	85
2006111121	04	87
2006111121	05	95
2009112001	01	79
2009112001	02	86
…	…	…

图 2-7 综合算例的基本关系

【例 2.3】 查询没有先行课的名称和学分

$$\pi_{Cname, Ccredit}(\sigma_{Cpno='NULL'}(C)) \text{ 或 } \pi_{2,4}(\sigma_{3='NULL'}(C))$$

运算结果为：

Cname	Ccredit
高等数学	6
计算机原理	4

【例 2.4】 查询选修了“02”课程的学生姓名

$$\pi_{Sname}(\pi_{Sname, Sno}(S) \bowtie \sigma_{Cno='02'}(SC))$$

运算结果为：

Sname
王昕
李彤

【例 2.5】 查询学号为 2006111121 的学生选修的课程名称和成绩

$$\pi_{Cname, Grade}(\pi_{Cno}(C) \bowtie \sigma_{Sno='2006111121'}(SC))$$

运算结果为：

Cname	Grade
程序设计基础	92
数据结构	85
计算机原理	87
数据库原理	95

【例 2.6】 查询选修了课程 02 的学生学号和姓名

$$\pi_{Sno, Sname}(\pi_{Sno, Sname}(S) \bowtie \sigma_{Cno='02'}(SC))$$

运算结果为：

Sno	Sname
2006111121	王昕
2009112001	李彤

【例 2.7】 查询选修了课程“数据结构”的学生姓名和所在班级

$$\pi_{Sname, Sclass}(\pi_{Sno, Sname, Sclass}(S) \bowtie \pi_{Sno,Cno}(SC) \bowtie \pi_{Cno}(\sigma_{Cname='数据结构'}(C)))$$

运算结果为：

Sname	Sclass
张宇	CS0701
王昕	SE0601

【例 2.8】 查询没有选修课程 02 的学生学号

$$\pi_{Sno}(S)-(\pi_{Sno}(\sigma_{Cno='02'}(SC)))$$

运算结果为：

Sno
2007111001
2007111002
2006111122
2009111001

【例 2.9】 查询选修了课程 02 和课程 03 的学生学号

$$\pi_{Sno,\ Cno}(SC)\div(\pi_{Cno}(\sigma_{Cno='02'\wedge Cno='03'}(C)))$$

运算结果为：

Sno
2006111121

【例 2.10】 查询选修了学生 2007111001 所选修课程的学生学号

$$\pi_{Sno,\ Cno}(SC)\div(\pi_{Cno}(\sigma_{Sno='2007111001'}(SC)))$$

运算结果为：

Sno
2006111121
2007111001

2.5 本章小结

本章系统地讲解了关系数据库的重要概念，并着重对关系模型进行了描述。关系模型包括关系数据结构、关系操作集合，以及关系完整性约束三个组成部分。本章还通过实例详细讲解了关系代数，包括传统的并、交、差、笛卡儿积集合操作和特殊的选择、投影、连接、除运算。

2.6 习题

2.6.1 名词解释

笛卡儿积、关系、元组、域、主码、属性、候选码、关系操作、关系模式、关系数据库

2.6.2 简答题

1. 简述关系数据库的特点。
2. 关系模型的完整性约束有哪几类？

3. 简述等值连接与自然连接的区别与联系。

2.6.3 综合题

已知一个“学生课程管理”数据库，其中有三个关系：

S(SNO, SNAME, CITY)——供应商(编号、名称、城市)

P(PNO, PNAME, COLOR, WEIGHT)——零件(编号、名称、颜色、重量)

J(JNO, JNAME, CITY)——工程(编号、名称、城市)

SPJ(SNO, PNO, JNO, QTY)——供应(供应商编号、零件编号、工程编号、数量)

请用关系代数表达式描述下列查询问题：

(1) 求供应工程 *J*1 零件的供应商号码 SNO；

(2) 求供应零件 *P*1 的供应商号码 SNO；

(3) 供应工程 *J*1 零件为红色的供应商号码 SNO；

(4) 求供应工程 *J*1 零件重量为 50 的供应商号码 SNO；

(5) 求供应工程 *J*1 零件为红色而且重量为 50 的供应商号码 SNO；

(6) 求至少使用了零件 *P*1 和零件 *P*2 的全部工程号 JNO；

(7) 求没有使用天津供应商生产的红色零件的工程号 JNO。

第3章 关系数据库标准语言SQL

结构化查询语言(Structured Query Language,SQL)是关系数据库的标准语言,其功能包括定义、查询、操纵和控制四个方面。

3.1 SQL简介

SQL是1974年由Boyce和Chamberlin提出的,其前身是SQUARE语言。1975—1979年,IBM公司的San Jose Research Laboratory研制出了著名的关系数据库管理系统原型System R,并实现了这种语言。SQL结构简洁,功能强大,简单易学,所以自从IBM公司于1981年推出以来,SQL得到了广泛的应用。如今无论是像Oracle、Sybase、Informix、SQL Server这些大型的数据库管理系统,还是像Visual Foxpro、PowerBuilder这些微机上常用的数据库开发系统,都支持以SQL作为查询语言。

3.1.1 SQL的特点

1. 综合统一

SQL集数据定义语言DDL、数据操纵语言DML、数据控制语言DCL的功能于一体,语言风格统一,可以独立完成数据库生命周期中的全部活动,包括定义关系模式、录入数据以建立数据库、查询、更新、维护、数据库重构、数据库安全性控制等一系列操作,这就为数据库应用系统开发提供了良好的环境。例如用户在数据库投入运行后,还可根据需要随时逐步地修改模式,并不影响数据库的运行,从而使系统具有良好的可扩展性。

2. 高度非过程化

非关系数据模型的数据操纵语言是面向过程的语言,用其完成某项请求,必须指定存取路径。而用SQL进行数据操作,用户只需提出"做什么",而不必指明"怎么做",因此用户无须了解存取路径,存取路径的选择以及SQL语句的操作过程由系统自动完成。这不但大大减轻了用户负担,而且有利于提高数据独立性。

3. 面向集合的操作方式

SQL采用集合操作方式,不仅查找结果可以是元组的集合,而且一次插入、删除、更新操作的对象也可以是元组的集合。

非关系数据模型采用的是面向记录的操作方式，任何一个操作其对象都是一条记录。例如查询所有平均成绩在80分以上的学生的姓名，用户必须说明完成该请求的具体处理过程，即如何用循环结构按照某条路径一条一条地把满足条件的学生记录读出来。

4. 以同一种语法结构提供两种操作方式

SQL既是自含式语言，又是嵌入式语言。作为自含式语言，它能够独立地用于联机交互的使用方式，用户可以在终端键盘上直接输入SQL命令对数据库进行操作；作为嵌入式语言，SQL语句能够嵌入到高级语言(C、COBOL、FORTRAN、PL/SQL等)程序中，供程序员设计程序时使用。虽然存在两种不同的使用方式，但SQL的语法结构基本是一致的。这种以统一语法结构提供两种不同使用方式的做法，为用户提供了极大的灵活性与方便性。

5. 语言简洁，易学易用

SQL功能极强，但由于设计巧妙，语言简洁，完成数据定义、数据操纵、数据控制的核心功能只用了9个动词：CREATE、DROP、ALTER、SELECT、INSERT、UPDATE、DELETE、GRANT、REVOKE，如表3-1所示。而且SQL语法简单，接近于英语口语，因此容易学习，容易使用。

表3-1 SQL的动词及其含义

SQL功能	SQL动词	SQL动词应用对象
数据定义	CREATE、DROP、ALTER	表、视图、索引
数据查询	SELECT	表、视图
数据操纵	INSERT、UPDATE、DELETE	表、视图
数据控制	GRANT、REVOKE	数据库、表

3.1.2 SQL语言简介

SQL语言同一般的计算机语言一样，也由以下几个部分组成。

1. 常量

SQL的常量一般包括文本常量(字符串)、整型常量、数值常量等。

2. 数据类型

SQL一般包括表3-2所列的数据类型。

表3-2 SQL的数据类型

数据类型	说明符	注释
整型	INT	定长16位
长整型	LONG	定长32位
十进制数	NUMBERIC(*m*, *n*)	*m*为十进制数，*n*为小数点位数
浮点数	FLOAT	定长64位(双精度)

续表

数据类型	说明符	注释
字符型(定长)	CHAR(n)	按固定长度 n 存储字符串，自动补充空格
变长字符型	VARCHAR(n)	按实际长度 n 存储字符串
日期型	DATE	格式为：yyyymmdd(年月日)
时间型	TIME	格式为：hhmmss(时分秒)

不同的数据库系统支持的数据类型不完全相同。例如，IBM DB2 SQL 主要支持表 3-3 所示的数据类型。

表 3-3　IBM DB2 SQL 主要支持的数据类型

数据类型	含义
SMALLINT	半字长二进制整数
INTEGERINT	全字长二进制整数
DECIMAL(p[,q])	压缩十进制数共 p 位，小数点后
或 DEC(p[,q])	$0 \leqslant q \leqslant p \leqslant 15$，$q=0$ 时可以省略
FLOAT	双字长浮点数
CHARTER(n)或 CHAR(n)	长度为 n 的定长字符串
VARCHAR(n)	最大长度为 n 的变长字符串
GRAPHIC(n)	长度为 n 的定长图形字符串
VARGRAPHIC(n)	最大长度为 n 的变长图形字符串
DATE	日期型，格式为 YYYY-MM-DD
TIME	时间型，格式为 HH. MM. SS
TIMESTAMP	日期加时间

3. 运算符

运算符列举如下：

- 算术运算符：+、-、*、/；
- 字符串运算符：||(连接)；
- 比较运算符：=、! =、<>、>、>=、<、<=；
- 逻辑运算符：NOT、AND、OR；
- 集合运算符：UNION、INTERSECT、MINUS。

4. 函数

SQL 提供了丰富函数，如聚集函数等。

SQL 的常量一般包括文本常量(字符串)、整型常量、数值常量等。一般函数引用形式：函数名([DISTINCT/ALL<表达式>])。主要的聚集函数将在后文中介绍。

5. 谓词

SQL 为了有更强的查询能力，提供了一系列的谓词，如 BETWEEN…END、IN、LIKE 等。

6. 表达式

表达式是指由一个或多个值、运算符和函数组合而成，并且可以计算出一个值的式子。

7. 条件

条件是指由一个或多个含有比较运算符及逻辑运算符组合而成的式子，其计算值是 T、F 或 NULL 的式子。

8. 命令

SQL 提供了丰富的数据处理命令，常用的有 CREATE、SELECT、INSERT、DELETE、UPDATE、GRANT 等。

3.2 SQL 数据定义功能

SQL 数据定义功能包括定义模式、定义表、定义索引和定义视图，其语句如表 3-4 所示。

表 3-4 SQL 的数据定义语句

SQL 功能	操作方式		
	创 建	删 除	修 改
模式	CREATE SCHEMA	DROP SCHEMA	无
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	无
索引	CREATE INDEX	DROP INDEX	无

3.2.1 创建、删除模式

1. 创建模式

创建模式是数据库中最基本的操作，其格式为：

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>
```

只有获得了授权的用户才能创建模式。

【例 3.1】 用户 Li 定义一个数据库模式“教学管理”EM

```
CREATE SCHEMA"EM"AUTHORIZATION Li
```

如果没有指定模式名，那么模式名隐含为用户名，请看下面的例子。

【例 3.2】 用户 Li 定义一个数据库模式“Li”

```
CREATE SCHEMA AUTHORIZATION Li
```

2. 删除模式

删除模式的格式为：

```
DROP SCHEMA <模式名> <CASCADE | RESTRICT>
```

其中 CASCADE 和 RESTRICT 必选其一。CASCADE(级联)表示在删除模式的同时把模式中所有的数据库对象全部一起删除掉；RESTRICT(限制)表示如果该模式中已经定义了下属的数据库对象(如表、视图等)，则拒绝执行删除模式，只有当模式中没有任何数据库对象时才能删除模式。

【例 3.3】 删除数据库模式“教学管理 EM”同时删除其中所有的数据库对象。

```
DROP SCHEMA"EM" CASCADE
```

3.2.2 创建、删除、修改基本表

1. 创建基本表

创建基本表是数据库中最基本的操作，其格式为：

```
CREATE TABLE <基本表名>(
    <列名 1> <数据类型> [列名 1 的列级完整性约束条件]
    [,<列名 2> <数据类型> [列名 2 的列级完整性约束条件]
    …
    ,<列名 n> <数据类型> [列名 n 的列级完整性约束条件]]
    [,<表级完整性约束条件 1>
        [,<表级完整性约束条件 2>]
        …
        [,<表级完整性约束条件 m>]
    ]
);
```

其中，<表名>是所要定义的基本表的名字，它可以由一个或多个属性(列)组成。每列由列名、数据类型和列级完整性约束条件组成，定义不同列时用逗号隔开。

建表的同时通常还可以定义若干与该表有关的完整性约束条件，这些完整性约束条件被存入系统的数据字典中，当用户操作表中数据时由 DBMS 自动检查该操作是否违背这些完整性约束条件。

【例 3.4】 建立一个“课程表”*C*，它由课程编号 Cno、名称 Cname、学分 Ccredit、前导课程编号 Cpno 四个属性组成。其中课程编号不能为空，值是唯一的，并且名称取值也是唯一的。

```
CREATE TABLE C
    ( Cno CHAR(3) NOT NULL UNIQUE,          /* 列级完整性约束条件,Cno 不空且唯一 */
      Cname CHAR(20) UNIQUE,                /* 列级完整性约束条件,Cname 唯一 */
      Cpno CHAR(3),
      Ccredit INT
    );
```

系统执行上面的 CREATE TABLE 语句后，就在数据库中建立了一个新的空的“课程表”*C*，并将有关“课程表”的定义及有关约束条件存放在数据字典中。

定义表的各个属性时需要指明其数据类型及长度。

【例 3.5】 建立一个"课程表"*C* 的同时先行课是自身的外码。

```
CREATE TABLE C
    ( Cno CHAR(3) PRIMARY KEY,                    /* 表级完整性约束条件 */
      Cname CHAR(20) UNIQUE,                      /* 表级完整性约束条件 */
      Cpno CHAR(3),
      Ccredit INT,
      FOREIGN KEY (Cpno) REFERENCES C(Cno)        /* 表级完整性约束条件 */
);
```

【例 3.6】 建立一个"学生表"*S* 和"选修表"*SC*。

```
CREATE TABLE S
    ( Sno CHAR(10) PRIMARY KEY,                   /* 列级完整性约束条件,Sno 是主码 */
      Ssex CHAR(1),
      Sage INT,
      Sclass CHAR(6)
    );
CREATE TABLE SC
    ( Sno CHAR(10),
      Cno CHAR(3),
      Grade INT,
      PRIMARY KEY( Sno, Cno),
      FOREIGN KEY(Sno)REFERENCES S(Sno),          /* 表级完整性约束条件 */
      FOREIGN KEY(Cno)REFERENCES C(Cno)           /* 表级完整性约束条件 */
    );
```

2. 修改基本表

修改已建立好的基本表,包括增加新列、增加新的完整性约束条件、修改原有的列定义或删除已有的完整性约束条件等,其一般格式为:

```
ALTER TABLE <表名>
    [ ADD <新列名> <数据类型> [完整性约束] ]
    [ DROP <完整性约束名>]
    [ MODIFY <列名> <数据类型>];
```

其中<表名>指定需要修改的基本表;ADD 子句用于增加新列和新的完整性约束条件;DROP 子句用于删除指定的完整性约束条件;MODIFY 子句用于修改原有的列定义。

【例 3.7】 向 *C* 表增加"课程类型"列,其数据类型为定长字符串。

```
ALTER TABLE C ADD Ctype CHAR(16);
```

不论基本表中原来是否已有数据,新增加的列一律为空值。

【例 3.8】 删除课程名称 Cname 必须取唯一值的约束。

```
ALTER TABLE C DROP UNIQUE Cname;
```

【例 3.9】 将成绩的数据类型改为浮点型。

```
ALTER TABLE SC MODIFY Grade FLOAT(4);
```

SQL没有提供删除属性列的语句,用户只能先将原表中要保留的列及其内容复制到一个新表中,然后删除原表,并将新表重命名为原表名。

3. 删除基本表

删除基本表的一般格式为:

```
DROP TABLE <表名> [<CASCADE | RESTRICT>];
```

缺省情况是RESTRICT,表示只有当要删除的基本表没有被其他基本表的约束(如外码)所引用、没有视图和触发器、存储过程或函数时,才能被删除。

CASCADE则表示,删除表时,与之相关的内容(视图、触发器等)一起被删除,因此应谨慎使用。

【例3.10】 删除*C*表。

```
DROP TABLE C;
```

注意:不同的数据库产品对CASCADE和RESTRICT的处理会有小小的差别,需要针对具体数据库做具体了解。

3.2.3 创建、删除、修改索引

建立索引是加快表的查询速度的有效手段。索引就像书的目录一样,可以直接定位到要查找的内容。SQL语言支持用户根据应用环境的需要,在基本表上建立一个或多个索引,以提供多种存取路径,加快查找速度。

1. 建立索引

建立索引的一般格式为:

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
      ON <表名> (<列名1> [<次序>][,<列名2> [<次序>]]…);
```

其中,<表名>指定要建索引的基本表的名字。索引可以建在该表的一列或多列上,各列名之间用逗号分隔。每个<列名>后面还可以用<次序>指定索引值的排列次序,包括ASC(升序)或DESC(降序),默认值为ASC。

UNIQUE为唯一性索引,表示此索引的每一个索引值只对应唯一的数据记录。

CLUSTER为聚簇索引,指索引项的顺序与表中记录的物理顺序一致的索引组织。例如CREATE CLUSTER INDEX Cname ON C(Cname);将会在表*C*的Cname列建一个聚簇索引,而且表中的记录将按照Cname值的升序存放。

用户可以在最常查询的列上建立聚簇索引以提高查询效率。显然在一个基本表上最多只能建立一个聚簇索引。建立聚簇索引后,更新索引列数据时,往往导致表中记录的物理顺序的变更,代价较大,因此对于经常更新的列不宜建立聚簇索引。

【例3.11】 为课程表*C*按课程号升序建立唯一性索引。

```
CREATE UNIQUE INDEX idxCno ON C(Cno);
```

【例 3.12】 为学生表 S 按学号升序建立聚簇索引。

```
CREATE CLUSTER INDEX CidxSno ON S(Sno);
```

2. 删除索引

删除索引的一般格式为：

```
DROP INDEX <索引名>;
```

【例 3.13】 删除 C 表的 idxCno 索引。

```
DROP INDEX idxCno;
```

删除索引时，系统会同时从数据字典中删去有关该索引的描述。

3.3 SQL 数据查询功能

SQL 语言提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。其一般格式为：

```
SELECT [ALL | DISTINCT]<目标列表达式>[,<目标列表达式>] …
FROM <表名或视图名>[,<表名或视图名>] …
[WHERE <条件表达式>]
[GROUP BY<列名 1 >[HAVING <条件表达式>]]
[ORDER BY<列名 2 >[ASC| DESC]];
```

SELECT 语句的含义是：根据 WHERE 子句的条件表达式，从 FROM 子句指定的基本表或视图中找出满足条件的元组，再按 SELECT 子句中的目标列表达式，选出元组中的属性值形成结果表。如果有 GROUP 子句，则将结果按＜列名 1＞的值进行分组，该属性列值相等的元组为一个组。通常会在每组中使用集函数。如果 GROUP 子句带 HAVING 短语，则只有满足指定条件的组才输出。如果有 ORDER 子句，则结果表还要按＜列名 2＞的值升序或降序排序。

在后面的例题中，将使用到图 3-1 所示的表。

3.3.1 单表查询

1. 选择表中的若干列

(1) 查询指定列

【例 3.14】 查询所有课程的编号和名称。

```
SELECT Cno, Cname
FROM C;
```

可根据应用的需要改变列的显示顺序。

(2) 查询全部列

【例 3.15】 查询所有课程的详细信息。

表 *S*

Sno	Sname	Ssex	Sage	Sclass
2007111001	张宇	M	20	CS0701
2007111002	赵娜	F	19	CS0701
2007111121	王昕	F	21	SE0601
2006111121	李伟	M	20	SE0601
2009111001	张莉	F	18	MA0901
2009112001	李彤	F	19	MA0901

表 *SC*

Sno	Cno	Grade
2007111001	03	64
2007111001	05	81
2007111001	06	72
2006111121	02	92
2006111121	03	85
2006111121	04	87
2006111121	05	95
2009112001	01	79
2009112001	02	86

表 *C*

Cno	Cname	Cpno	Ccredit
01	高等数学		6
02	程序设计基础	01	6
03	数据结构	02	6
04	计算机原理		4
05	数据库原理	03	3
06	OO 程序设计	01	4
07	操作系统	03	4
08	计算机网络	07	3

图 3-1 查询算例的基本表

```
SELECT *
FROM C;
```

(3) 查询经过计算的值

【例 3.16】 查全体学生的姓名及其出生年份。

```
SELECT Sname, 2011 - Sage
FROM S;
```

查询结果为：

Sname	2011-Sage	Sname	2011-Sage
张宇	1991	李伟	1991
赵娜	1992	张莉	1993
王昕	1990	李彤	1992

<目标列表达式>还可以是字符串常量、函数等，而且可以通过指定别名来改变查询结果的列标题。

【例 3.17】 查询全体学生的姓名、出生年份和所在系，要求用小写字母表示所有系名。

```
SELECT Sname NAME, 2011 - Sage BIRTHDATE, LOWER(Sclass) CLASS
FROM S;
```

查询结果为：

NAME	BIRTHDATE	CLASS
张宇	1991	cs0701
赵娜	1992	cs0701
王昕	1990	se0601
李伟	1991	se0601
张莉	1993	ma0901
李彤	1992	ma0901

2. 选择表中的若干元组

(1) 消除取值重复的行

两个本来并不完全相同的元组，投影到指定的某些列上后，可能变成相同的行了。例如查询选修了课程的学生学号，因为某个学生选择了若干门课，因此他的学号会重复出现，如果想去掉结果表中的重复行，就必须指定 DISTINCT 短语。

【例 3.18】 查询选修了课程的学生学号。

```
SELECT DISTINCT Sno
FROM SC;
```

如果未指定 DISTINCT 短语，则缺省为 ALL，即保留结果表中取值重复的行。

(2) 查询满足条件的元组

查询满足指定条件的元组可以通过 WHERE 子句来实现。WHERE 子句常用的查询条件，如表 3-5 所示。

表 3-5 常用的查询条件

查询条件	谓词
比较	=，>，<，<=，>=，<>，!>，!<
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空值	IS NULL，IS NOT NULL
多重条件	AND，OR

① 比较大小

【例 3.19】 查询全体男生的名单。

```
SELECT Sname
FROM S
WHERE Ssex = 'M';
```

② 确定范围

【例 3.20】 查询所有年龄在 19～22 岁(包括 19 岁和 22 岁)的学生姓名及其年龄。

```
SELECT Sname, Sage
FROM S
WHERE Sage BETWEEN 19 AND 22;
```

③ 确定集合

【例 3.21】 查询除了数据结构、计算机原理、数据库原理以外的课程编号和课程名。

```
SELECT Cno, Cname
FROM C
WHERE Cname NOT IN ('数据结构','计算机原理','数据库原理');
```

④ 字符匹配

谓词 LIKE 可以用来进行字符串的匹配，其格式为：

```
[NOT] LIKE '<匹配串>'[ESCAPE'<换码字符>']
```

其含义是查找指定的属性列值与<匹配串>相匹配的元组。<匹配串>可以为完整的字符串，也可以含有通配符%和_。其中：%（百分号）代表任意长度（长度可以为 0）的字符串；_代表任意单个字符。例如 a%b 表示以 a 开头，以 b 结尾的任意长度的字符串，asb、asdb、aefegb 均满足该匹配串；a_b 表示以 a 开头，以 b 结尾的长度为 3 的任意字符串，acb、asb、azb 满足该匹配串，而 asdb、aefegb 不满足该匹配串。

【例 3.22】 查询所有姓张的学生的姓名、学号和性别。

```
SELECT Sname,Sno, Ssex
FROM S
WHERE Sname LIKE '张%';
```

【例 3.23】 查询姓名中第二个字为"娜"的学生的姓名和学号。

```
SELECT Sname,Sno
FROM S
WHERE Sname LIKE '_娜%';
```

⑤ 涉及空值的查询

【例 3.24】 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。查询有成绩的学生的学号和相应的课程号、成绩。

```
SELECT *
FROM  SC
WHERE GRADE IS NOT NULL;
```

⑥ 多重条件

【例 3.25】 查询所有年龄在 20 岁以上（包括 20 岁）的男学生姓名及其年龄。

```
SELECT Sname, Sage FROM S WHERE Sage<=20 AND Ssex='M';
```

3. 对查询结果排序

【例 3.26】 查询选修了 03 号课程的学生的选课记录，查询结果按分数的降序排列。

```
SELECT *
FROM SC
WHERE Cno='03'
ORDER BY Grade DESC;
```

【例 3.27】 查询全体学生情况，查询结果按所在班级的升序排列，同一班的学生按年龄降序排列。

```
SELECT *
FROM S
ORDER BY Sclass, Sage DESC;
```

4. 使用集函数

为了方便用户查询，SQL 提供了一些集函数，如表 3-6 所示。

表 3-6 SQL 提供的集函数

集　函　数	含　　义
COUNT(*)	统计元组个数
COUNT([DISTINCT | ALL]＜列名＞)	统计一列中值的个数
SUM([DISTINCT | ALL]＜列名＞)	计算一列值的总和(此必为数值)
AVG([DISTINCT | ALL]＜列名＞)	计算一列值的平均值(此必为数值)
MAX([DISTINCT | ALL]＜列名＞)	求一列值中的最大值
MIN([DISTINCT | ALL]＜列名＞)	求一列值中的最小值

【例 3.28】 统计课程总数。

```
SELECT COUNT (*)
FROM C;
```

【例 3.29】 统计被学生选择的课程总数。

```
SELECT COUNT (DISTINCT SNO )
FROM SC;
```

【例 3.30】 计算选修 01 号课程的学生的平均成绩。

```
SELECT AVG(Grade)
FROM SC
WHERE Cno = '01';
```

【例 3.31】 查询选修 01 号课程学生的最高分数。

```
SELECT MAX(Grade)
FROM SC
WHERE Cno = '01';
```

5. 对查询结果分组

GROUP BY 子句可以将查询结果表的各行按一列或多列取值相等的原则进行分组。

对查询结果分组的目的是细化集函数的作用对象。如果未对查询结果分组，集函数将作用于整个查询结果，即整个查询结果只有一个函数值，如例 3.28～例 3.31。否则，如果对查询结果分组，集函数将作用于每一个组，即每一组都有一个函数值。

【例 3.32】 求每门课程的平均分。

```
SELECT Cno, AVG(Grade)
FROM SC
GROUP BY Cno;
```

【例 3.33】 查询学生选修课程超过 3 人的课程编号。

```
SELECT Cno
FROM SC
GROUP BY Cno
HAVING COUNT( * )>= 3;
```

注意：WHERE 与 HAVING 有区别，它们的作用对象不同。WHERE 作用于基本表或视图，从中选择满足条件的元组；HAVING 作用于组，从中选择满足条件的组。

3.3.2 连接查询

若一个查询同时涉及两个以上的表，则称之为连接查询。连接查询实际上是关系数据库中最主要的查询，主要包括等值连接、非等值连接、自然连接、自身连接、外连接和复合条件连接查询。

1. 等值与非等值连接查询

连接条件写在 WHERE 子句中，格式为：

```
[表名 1.]<列名 1><比较运算符> [表名 2.] <列名 2>
```

比较运算符包括=、! =、>、>=、<、<=。如果比较运算符为=，则称为等值连接，否则称为非等值连接。

【例 3.34】 查询每个学生及其选修课程的情况。

```
SELECT S. * , SC. *
FROM S, SC
WHERE S.Sno = SC.Sno;
```

该查询的执行结果为：

S. Sno	Sname	Ssex	Sage	Sclass	SC. Sno	Cno	Grade
2007111001	张宇	M	20	CS0701	2007111001	3	64
2007111001	张宇	M	20	CS0701	2007111001	5	81
2007111001	张宇	M	20	CS0701	2007111001	6	72
2006111121	王昕	F	21	SE0601	2006111121	2	92
2006111121	王昕	F	21	SE0601	2006111121	3	85
2006111121	王昕	F	21	SE0601	2006111121	5	87
2006111121	王昕	F	21	SE0601	2006111121	5	95
2009112001	李彤	F	19	MA0901	2009112001	1	79
2009112001	李彤	F	19	MA0901	2009112001	2	86

连接运算有两种特殊情况：广义笛卡儿积连接和自然连接。

两个表的广义笛卡儿积连接为两表中元组的交叉乘积，结果会产生一些没有意义的元

组,实际很少使用。若在等值连接中把目标列中重复的属性列去掉则为自然连接。例 3.34 可用自然连接完成如下:

```
SELECT S.Sno, Sname, Ssex, Sage, Sclass, Cno, Grade
FROM S, SC
WHERE S.Sno = SC.Sno;
```

2. 自身连接

连接操作不仅可以在两个表之间进行,也可以是一个表与自己进行连接,称为表的自身连接。

【例 3.35】 查询每一门课的间接先行课(即先行课的先行课)。

为 *C* 表取两个别名,FIRST、SECOND,完成该查询的 SQL 语句为:

```
SELECT FIRST.Cno, SECOND.Cpno
FROM C FIRST, C SECOND
WHERE FIRST.Cpno = SECOND.Cno AND SECOND.CPno IS NOT NULL;
```

该查询的执行结果为:

Cno	Cpno	Cno	Cpno
03	01	07	02
05	02	08	03

3. 外连接

在通常的连接操作中,只有满足连接条件的元组才能作为结果输出。如例 3.34 的结果表中没有学生 2007111002 的信息。如果想以 S 表为主体,列出每个学生的基本情况及其选课情况,若某个学生没有选课,只输出其基本情况信息,其选课信息为空值即可,就需要使用外连接(OUTER JOIN)。可以如下改写例 3.34:

```
SELECT S.Sno, Sname, Ssex, Sage, Sclass, Cno, Grade
FROM S LEFT OUTER JOIN SC
ON S.Sno = SC.Sno;
```

外连接的表示方法为:加入关键字"表 1 LEFT/RIGHT OUTER JOIN 表 2 ON 谓词表达式。"

外连接就好像是为符号 * 所在边的表增加一个"万能"的行,这个行全部由空值组成,它可以和另一边的表中所有不满足连接条件的元组进行连接。

执行结果如下:

Sno	Sname	Ssex	Sage	Sclass	Cno	Grade
2007111001	张宇	M	20	CS0701	06	72
2007111001	张宇	M	20	CS0701	05	81
2007111001	张宇	M	20	CS0701	03	64
2007111002	赵娜	F	19	CS0701	NULL	NULL
2006111121	王昕	F	21	SE0601	05	95

续表

Sno	Sname	Ssex	Sage	Sclass	Cno	Grade
2006111121	王昕	F	21	SE0601	05	87
2006111121	王昕	F	21	SE0601	03	85
2006111121	王昕	F	21	SE0601	02	92
2006111122	李伟	M	20	SE0601	NULL	NULL
2009111001	张莉	F	18	MA0901	NULL	NULL
2009112001	李彤	F	19	MA0901	02	86
2009112001	李彤	F	19	MA0901	01	79

4. 复合条件连接

WHERE 子句中可以有多个连接条件，称为复合连接条件。

【例 3.36】 查询选修 02 号课程且成绩在 90 分以上的所有学生。

```
SELECT S.Sno, Sname
FROM S, SC
WHERE S.Sno = SC.Sno AND SC.Cno = '02' AND SC.Grade > 90;
```

连接操作除了可以是两表连接或一个表自身连接外，还可以是两个以上的表进行连接，即多表连接。

【例 3.37】 查询每个学生的学号、姓名、选修的课程及成绩。

```
SELECT S.Sno,Sname, Cname, Grade
FROM S, SC, C
WHERE S.Sno = SC.Sno AND SC.Cno = C.Cno;
```

3.3.3 嵌套查询

在 SQL 中，一个 SELECT-FROM-WHERE 称为一个查询块。将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询称为嵌套查询。

【例 3.38】 查询学号为“2006111121”的学生选修的课程名称。

```
SELECT Cname
FROM C
WHERE Cno IN (SELECT Cno
FROM SC
WHERE Sno = '2006111121');
```

上层的查询块称为外层查询或父查询，下层查询块称为内层查询或子查询。例 3.38 中，SELECT Cno FROM SC WHERE Sno='2006111121'为子查询。SQL 允许多层嵌套查询。另外，子查询的 SELECT 语句中不能使用 ORDER BY 子句，ORDER BY 子句只能对最终查询结果排序。

嵌套查询一般的求解方法是由里向外处理，即每个子查询在上一级查询处理之前求解。子查询的结果用于建立其父查询的查找条件。

嵌套查询使多个简单查询构成了复杂查询，从而增强了 SQL 的查询能力。

1. 带有 IN 谓词的子查询

【例 3.39】 查询以“程序设计基础”为先行课的课程。

分步查询：

① 确定“程序设计基础”的课程编号；

```
SELECT Cno
FROM C
WHERE Cname = '程序设计基础';
```

结果为：'02'

② 查找所有以'02'为先行课的课程。

```
SELECT *
FROM C
WHERE Cpno = '02';
```

使用 IN 的嵌套查询：

```
SELECT *
FROM C
WHERE Cpno IN
(SELECT Cno
FROM C
WHERE Cname = '程序设计基础');
```

结果为：

Cno	Cname	Cpno	Ccredit
03	数据结构	02	6

DBMS 求解嵌套时也是分步执行的，类似于分步查询，即首先确定“程序设计基础”的课程编号，然后求解父查询，查找所有以'03'为先行课的课程。

本例中的查询也可以用自身连接来完成：

```
SELECT FIRST. *
FROM C FIRST, C SECOND
WHERE SECOND.Cname = '程序设计基础' AND FIRST.Cpno =  SECOND.Cno;
```

本例中父查询和子查询均引用了 C 表，可以像自身连接那样用别名将父查询中的 C 表与子查询中的 C 表区分开：

```
SELECT *
FROM C C1
WHERE C1.Cpno IN
          (SELECT Cno FROM C C2 WHERE C2.Cname = '程序设计基础');
```

【例 3.40】 查询选修了课程名为“程序设计基础”的学生。

```
SELECT *
FROM S
WHERE Sno IN
          (SELECT Sno
           FROM SC
           WHERE Cno IN
                     (SELECT Cno FROM C WHERE Cname = '程序设计基础'));
```

结果为：

Sno	Sname	Ssex	Sage	Sclass
2006111121	王昕	F	21	SE0601
2009112001	李彤	F	19	MA0901

本查询同样可以用连接查询来实现：

```
SELECT S.*
FROM S, SC, C
WHERE S.Sno = SC.Sno AND SC.Cno = C.Cno AND
C.Cname = '程序设计基础';
```

查询涉及多个关系时，用嵌套查询逐步求解，层次清楚，易于构造，具有结构化程序设计的优点。

例 3.39 和例 3.40 中的各个子查询都只执行一次，其结果用于父查询。子查询的查询条件不依赖于父查询，这类子查询称为不相关子查询。不相关子查询是最简单的一类子查询。

2. 带有比较运算符的子查询

带有比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时，可以用>、<、=、>=、<=、!=或<>等比较运算符。例如：

```
SELECT Sno, Sname, Sclass
FROM S
WHERE Sclass =
             (SELECT Sclass
              FROM S
              WHERE Sname = '王昕');
```

注意：子查询一定要跟在比较运算符之后，下面的 SQL 语句是错误的：

```
SELECT Sno, Sname, Sclass
FROM S
WHERE (SELECT Sclass
       FROM S
       WHERE Sname = '王昕') = Sclass;
```

3.3.4 集合查询

SELECT 语句的查询结果是元组的集合，所以多个 SELECT 语句的结果可进行集合操

作。集合操作主要包括并操作 UNION、交操作 INTERSECT 和差操作 MINUS。

【例 3.41】 查询计科 0701 班的学生及年龄小于 20 岁的学生。

```
SELECT *
FROM S
WHERE Sclass = 'CS0701'
UNION
SELECT *
FROM S
WHERE Sage < 20;
```

使用 UNION 将多个查询结果合并起来时，系统会自动去掉重复元组。

注意：参加 UNION 操作的各结果表的列数必须相同；对应项的数据类型也必须相同。

标准 SQL 中没有直接提供集合交操作和集合差操作，但可以用其他方法来实现。如例 3.41 可以用下面的 SELECT 语句来实现：

```
SELECT * FROM S WHERE Sclass like 'CS%' OR Sage < 20;
```

【例 3.42】 查询选修课程 03 的学生集合与选修课程 05 的学生集合的交集。

```
SELECT Sno
FROM SC
WHERE Cno = '03'
INTERSECT
SELECT Sno
FROM SC
WHERE Cno = '05';
```

可以用下面的 SELECT 语句实现：

```
SELECT Sno
FROM SC
WHERE Cno = '03' AND Sno IN
     (SELECT Sno
     FROM SC
     WHERE Cno = '05');
```

【例 3.43】 查询计科 0701 班的学生与年龄小于 20 岁的学生的差集。

本查询换种说法就是，查询计科 0701 班中年龄大于 20 岁(含 20 岁)的学生。

```
SELECT *
FROM S
WHERE Sclass = 'CS0701'AND Sage > 20;
```

在使用 SELECT 语句进行查询时，要注意以下几点：

(1) SQL 中表名的顺序，条件的顺序无所谓，可以任意调换。

(2) WHERE 子句中的查询条件包括连接条件和选择条件。

(3) 子查询可以转化为多表的连接查询，而连接查询不一定能用子查询实现。

3.4 SQL 数据操纵功能

SQL 数据操纵功能包括对基本表和视图的操纵，本节将详细介绍对基本表的数据操纵功能，而对视图数据的操纵功能将在 3.5 节介绍。

SQL 中，对基本表的数据操纵功能是指对基本表中数据的插入、修改和删除。

3.4.1 插入数据

SQL 的数据插入语句 INSERT 有两种形式：一种是插入一个元组，另一种是插入子查询结果。

1. 插入单个元组

INSERT 语句的格式为：

```
INSERT
INTO <表名>[(<列名>,[<列名>…])]
VALUES (<常量>[,<常量>]…);
```

其功能为将 VALUES 后面的数据插入到指定表中。如果 INTO 子句中没有指明任何列名，则新插入的记录必须在每个属性列上均有值。如果 INTO 子句指明了列名，则新记录在未出现的属性列上取空值，定义时指明为 NOT NULL 的属性列不能为空值。

【例 3.44】 将一个新课程记录插入到 *C* 表中。

```
INSERT
INTO C
VALUES ('09','计算机图形学','', 3);
```

【例 3.45】 插入一条选课记录。

```
INSERT
INTO SC(Sno,Cno)
VALUES ( '2007111001','09');
```

2. 插入子查询结果

子查询不仅可以嵌套在 SELECT 语句中，用以构造父查询的条件，也可以嵌套在 INSERT 语句中，用以生成要插入的批量数据。

插入子查询结果的 INSERT 语句的格式为：

```
INSERT
INTO <表名>[(<列名>,[<列名>…])]
子查询;
```

【例 3.46】 对每一个班，求学生的平均年龄，并把结果存入数据库。

① 先建立一新表 Classage；

```
CREATE TABLE Classage(
Sclass CHAR(15),
Avgage SMALLINT);
```

② 按班分组求出平均年龄,再将系名和平均年龄插入新表 Deptage。

```
INSERT INTO Classage (Sclass, Avgage)
SELECT Sclass, AVG(Sage) FROM S
GROUP BY Sclass;
```

3.4.2 修改数据

修改数据语句的一般格式为:

```
UPDATE 表名
SET <列名>=<表达式>[,<列名>=<表达式>]…
[WHERE 条件];
```

其功能为修改指定表中满足 WHERE 子句条件的元组。其中 SET 子句给出<表达式>的值用于取代相应的属性列值。如果省略 WHERE 子句,则表示要修改表中的所有元组。

1. 修改某一个元组的值

【例 3.47】 将学生 2007111001 的年龄改为 21 岁。

```
UPDATE S
SET Sage = 21
WHERE Sno = '2007111001';
```

2. 修改多个元组的值

【例 3.48】 将所有学生的年龄增加 1 岁。

```
UPDATE S
SET Sage =  Sage + 1;
```

3. 带子查询的修改语句

子查询也可以嵌套在 UPDATE 语句中,用以构造修改的条件。

【例 3.49】 将计科 0701 班全体学生的成绩置零。

```
UPDATE SC
SET Grade = 0
WHERE Sno in(
            Select SC.Sno from SC,S
            WHERE SC.Sno = S.Sno AND Sclass = 'CS0701');
```

3.4.3 删除数据

删除数据语句的一般格式为:

```
DELETE
FROM <表名>
[WHERE <条件>];
```

其功能为从指定表中删除满足 WHERE 子句条件的所有元组。如果省略 WHERE 子句,则表示删除表中全部元组。

1. 删除某个元组的值

【例 3.50】 删除课程号为'02'的课程记录。

```
DELETE
FROM C
WHERE Cno = '02';
```

2. 删除多个元组的值

【例 3.51】 删除所有选课记录。

```
DELETE
FROM SC;
```

3. 带子查询的删除语句

子查询同样也可以嵌套在 DELETE 语句中,用以构造执行删除操作的条件。

【例 3.52】 删除所有课程名称为"计算机图形学"的选课记录。

```
DELETE FROM SC
WHERE Cno IN
      ( SELECT SC.Cno
       FROM C,SC
       WHERE Cname = '计算机图形学' AND SC.Cno = C.Cno
      );
```

4. 更新操作与数据库的一致性

增删改操作只能对一个表操作,这会带来一些问题。例如,学生 2007111001 被删除后,有关其选课信息也应同时删除。

① 删除学生 2007111001:

```
DELETE
FROM S
WHERE Sno = '2007111001';
```

② 删除学生 2007111001 的选课记录:

```
DELETE
FROM SC
WHERE Sno = '2007111001';
```

3.5 视图

视图是从一个或几个基本表(或视图)导出的表,与基本表不同,视图是一个虚表。数据库中只存放视图的定义,而不存放视图对应的数据,这些数据仍存放在原来的基本表中。所以基本表中的数据发生变化,从视图中查询出的数据也就随之改变了。

视图一经定义,就可以和基本表一样被查询、被删除,可以在一个视图之上再定义新的视图。

3.5.1 定义和删除视图

1. 建立视图

```
CREATE VIEW <视图名> [(<列名>[,<列名>]…)]
AS <子查询>
[WITH CHECK OPTION];
```

其中,子查询可以是任意复杂的 SELECT 语句,但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION 表示对视图进行 INSERT、UPDATE 和 DELETE 操作时要保证插入、修改或删除的记录满足视图定义中的谓词条件(即子查询中的条件表达式)。

组成视图的属性列名或者全部省略或者全部指定。如果省略了视图的各个属性列名,则隐含该视图由子查询中 SELECT 子句目标列中的所有字段组成。但在下列三种情况下必须明确指定组成视图的所有列名:

- 某个目标列不是单纯的属性名,而是集函数或列表达式。
- 多表连接时选出了几个同名列作为视图的字段。
- 需要在视图中为某属性列使用新的更合适的名字。

(1) 行列子集视图

若一个视图是从单个基本表导出的,并且只是去掉了基本表的某些行和某些列,但保留了码,则称这类视图为行列子集视图。

【例 3.53】 建立计科 0701 班学生的视图,并要求在进行修改和插入操作时仍需保证该视图中只有计算机系的学生。

```
CREATE VIEW CS_S
   AS
     SELECT Sno,Sname,Sage
     FROM S
     WHERE Sclass = 'CS0701'
     WITH CHECK OPTION;
```

DBMS 执行 CREATE VIEW 语句的结果只是把视图的定义存入数据字典,并不执行其中的 SELECT 语句。只是在对视图查询时,才按视图的定义从基本表中将数据查出。

(2)由视图导出的视图

视图不仅可以建立在一个或多个基本表上,还可以建立在一个或多个已经定义的视图上。

【例 3.54】 建立计科 0701 班选修了 03 号课程的学生的视图。

```
CREATE VIEW CS_S1(Sno, Sname, Grade)
   AS
      SELECT S.Sno, Sname, Grade
      FROM CS_S,SC
      WHERE Sclass = 'CS0701' AND CS_S.Sno = SC.Sno AND SC.Cno = '03';
```

(3) 带虚拟列的视图

定义基本表时,为了减少数据库中的冗余数据,表中只存放基本数据,由基本数据经过各种计算派生出的数据一般是不存储的。但由于视图中的数据并不实际存储,因此定义视图时可以根据应用的需要,设置一些派生属性列。这些派生属性列由于在基本表中并不实际存在,因此也称它们为虚拟列。带虚拟列的视图也称为带表达式的视图。

【例 3.55】 定义一个反映学生出生年份的视图。

```
CREATE VIEW BS(Sno, Sname, Sbirth)
      AS
          SELECT Sno,Sname,2011 - Sage
          FROM S;
```

BS 视图是一个带表达式的视图。视图中的出生年份值是通过计算得到的。

(4) 带有集函数的视图

还可以用带有集函数和 GROUP BY 子句的查询来定义视图,这种视图称为分组视图。

【例 3.56】 将学生的学号及他们的平均成绩定义为一个视图。

```
CREATE VIEW S_G(Sno,Gavg)
    AS
      SELECT Sno, AVG(Grade)
      FROM SC
      GROUP BY Sno;
```

2. 删除视图

格式:

```
DROP VIEW <视图名>;
```

视图删除后视图的定义将从数据字典中删除,但由该视图导出的其他视图定义仍在数据字典中,但已经失效,不能继续使用,应该将它们一一删除。

【例 3.57】 若删除例 3.53 中的视图 CS_S 就应该删除例 3.54 中的视图 CS_S1。

```
DROP VIEW CS_S1;
DROP VIEW CS_S;
```

3.5.2 查询视图

视图定义后,用户就可以像对基本表一样对视图进行查询了。

【例 3.58】 在计科 0701 班学生的视图中找出年龄小于 20 岁的学生。

```
SELECT Sno, Sage FROM CS_S WHERE Sage<20;
```

DBMS执行对视图的查询时，首先进行有效性检查，检查查询的表、视图等是否存在。如果存在，则从数据字典中取出视图的定义，把定义中的子查询和用户的查询结合起来，转换成等价的对基本表的查询，然后再执行修正后的查询。这一转换过程称为视图消解。

【例 3.59】 转换后的查询语句为：

```
SELECT Sno Sage FROM S WHERE Sclass='CS0701' AND Sage<20;
```

在一般情况下，视图查询的转换是直截了当的。但在有些情况下，这种转换不能直接进行，查询时就会出现问题。

【例 3.60】 在 S_G 视图中查询平均成绩在 90 分以上的学生学号和平均成绩。

```
SELECT *
FROM S_G
WHERE Gavg>=90;
```

将上面查询语句与子查询结合，形成下列查询语句。

```
SELECT Sno, AVG(Grade)
FROM SC
WHERE AVG(Grade)>=90
GROUP BY Sno;
```

前面讲过分组后 WHERE 子句中不能用集函数作为条件表达式，因此执行此修正后的查询将会出现语法错误。正确的查询语句应该是：

```
SELECT Sno,AVG(Grade)
FROM SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```

目前，多数关系数据库系统对行列子集视图的查询均能进行正确转换，但对非行列子集的查询就不一定能做转换了，因此这类查询应该直接对基本表进行。

3.5.3 更新视图

由于视图是不实际存储数据的虚表，因此对视图的更新最终要转换为对基本表的更新。为防止用户通过视图对数据进行增加、删除、修改时，有意无意地对不属于视图范围内的基本表数据进行操作，可在定义视图时加上 WITH CHECK OPTION 子句。这样在视图上增删改数据时，DBMS 会检查视图定义中的条件，若不满足条件，则拒绝执行该操作。

1. 插入视图数据

【例 3.61】 将计科 0701 班学生视图 CS_S 中学号为 2007111001 的学生的姓名改为“张强”。

```
UPDATE CS_S
SET Sname='张强'
WHERE Sno='2007111001';
```

转换后的更新语句为：

```
UPDATE S
SET Sname = '张强'
WHERE Sno = '2007111001' AND Sclass = 'CS0701';
```

2. 删除视图数据

【例 3.62】 向计科 0701 班学生视图 CS_S 中插入一个新的学生记录。

```
INSERT INTO CS_S
VALUES('2007111003','王欢',20);
```

转换为对基本表的更新：

```
INSERT INTO S(Sno, Sname, Sage, Sclass)
VALUES('2007111003', '王欢', 20,'CS0701');
系统自动将班级名'CS0701'放入 VALUES 子句中.
```

3. 修改视图数据

【例 3.63】 删除计科 0701 班学生视图 CS_S 中学号为 2007111003 的记录。

```
DELETE FROM CS_S
WHERE Sno = '2007111003';
```

转换为对基本表的更新：

```
DELETE FROM S
WHERE Sno = '2007111003' AND Sclass = 'CS0701';
```

在关系数据库中，并不是所有的视图都是可更新的，因为有些视图的更新不能唯一地有意义地转换成对相应基本表的更新。

例如前面定义的视图 S_G 是由“学号”和“平均成绩”两个属性列组成的，其中平均成绩一项是由 S 表中对元组分组后计算平均值得来的。如果想把视图 S_G 中学号为 2007111001 的学生的平均成绩改成 90 分，则 SQL 语句如下：

```
UPDATE S_G
SET Gavg = 90
WHERE Sno = '2007111001';
```

但这个对视图的更新是无法转换成对基本表的更新的。因为系统无法修改各科成绩，以使平均成绩为 90，所以 S_G 视图是不可更新的。一般地，行列子集视图是可更新的，除行列子集视图外，还有一些视图理论上是可更新的。

3.6 数据控制

数据控制也称为数据保护，包括数据的安全性控制、完整性控制、并发控制和恢复。关于数据保护的概念将在第 4 章中详细介绍，本节主要介绍 SQL 的数据控制功能。

数据库的安全性是指保护数据库，防止不合法的使用所造成的数据泄露和破坏。数据库系统中保证数据安全性的主要措施是进行存取控制，即规定不同用户对不同数据对象所允许的执行操作，并规定用户只能存取其权限内的数据。

3.6.1 授权

SQL 使用 GRANT 语句授予用户权限，其一般格式为：

```
GRANT <权限>[,<权限>]…
      [ON <对象类型><对象名>]
      TO <用户>[,<用户>]…
      [WITH GRANT OPTION];
```

其功能为将指定操作对象的指定操作权限授予指定的用户。

不同类型的操作对象有不同的操作权限，如表 3-7 所示。

接受权限的用户可以是一个或多个具体用户，也可以是 PUBLIC(表示全体用户)。

如果指定了 WITH GRANT OPTION 子句，则获得某种权限的用户还可以把这种权限再授予别的用户；如果没有指定 WITH GRANT OPTION 子句，则获得某种权限的用户只能使用该权限，但不能传播该权限。

【例 3.64】 把对 *C* 表和 *S* 表的全部权限授予用户 *U*1 和 *U*2。

```
GRANT ALL PRIVIEGES ON TABLE C, S TO U1,U2;
```

表 3-7 不同类型操作对象的操作权限

操作对象	对象类型	操作权限
属性列	TABLE	SELECT、INSERT、UPDATE、DELETE、ALL PRIVIEGES
视图	TABLE	SELECT、INSERT、UPDATE、DELETE、ALL PRIVIEGES
基本表	TABLE	SELECT、INSERT、UPDATE、DELETE ALTER、INDEX、ALL PRIVIEGES
数据库	DATABASE	CREATETAB

【例 3.65】 把查询 *C* 表和修改 Cname 的权限授予用户 *U*3，并允许他将此权限授予其他用户。

```
GRANT SELECT, UPDATE(Cname) ON TABLE C TO U3
     WITH GRANT OPTION;
```

【例 3.66】 把在数据库 *S* 中建立表的权限授予用户 *U*3。

```
GRANT CREATETAB ON DATABASE S TO U3;
```

注意：因为对象类型的关键字不同，所以数据库和表的权限授予要分开进行。

3.6.2 收回权限

授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回，其一般格式为：

```
REVOKE <权限>[,<权限>]…
        [ON <对象类型><对象名>]
        FROM <用户>[,<用户>]…;
```

【例 3.67】 把用户 $U1$ 对 C 表和 S 表的全部权限收回。

```
REVOKE ALL PRIVIEGES ON TABLE C, S FROM U1;
```

【例 3.68】 把用户 $U3$ 的修改 Cname 的权限收回。

```
REVOKE UPDATE(Cname) ON TABLE C FROM U3;
```

如果在收回 $U3$ 的权限之前，$U3$ 已经将此权限授予了 $U4$ 和 $U5$，则收回 $U3$ 权限的同时，$U4$ 和 $U5$ 的权限也自动级联收回了。如果 $U4$ 还从其他用户和 DBA 处获得了此权限，则此权限不能被收回。系统只收回直接或间接从 $U3$ 处获得的权限。

DBA 拥有对数据库中所有对象的所有权限，并可以根据应用的需要将不同的权限授予不同的用户。而所有授予出去的权限在必要时又都可以用 REVOKE 语句收回。

3.7 本章小结

本章讨论关系数据库标准语言 SQL。SQL 是关系数据库的标准语言，已广泛应用在商用系统中。首先分析了 SQL 语言的特点，然后通过一些实例分别对 SQL 语言的数据定义、数据查询和数据操纵功能做了详细讲解，涵盖了表的创建、表的简单查询与复杂查询、数据更新等常用操作。其中最为复杂的是 SELECT 查询语句，包括单表查询、连接查询、嵌套查询和集合查询几大类。并在此基础上介绍了视图的概念以及视图的定义、删除、查询和更新操作。最后介绍了用于数据控制的授权和收回权限的语句用法。

3.8 习题

3.8.1 名词解释

基本表、视图

3.8.2 简答题

1. 简述基本表和视图的区别和联系，试述 SQL 语言的特点。
2. 简述视图的优点。
3. 哪些视图是可以更新的？哪些视图是不可更新的？各举一例说明。

3.8.3 综合题

设有一个 SPJ 数据库，包括 S、P、J、SPJ 四个关系模式：

S(SNO,SNAME,STATUS,CITY)；

P(PNO,PNAME,COLOR,WEIGHT);

J(JNO,JNAME,CITY);

SPJ(SNO,PNO,JNO,QTY);

供应商表 S 由供应商代码(SNO)、供应商姓名(SNAME)、供应商状态(STATUS)、供应商所在城市(CITY)组成;

零件表 P 由零件代码(PNO)、零件名(PNAME)、颜色(COLOR)、重量(WEIGHT)组成;

工程项目表 J 由工程项目代码(JNO)、工程项目名(JNAME)、工程项目所在城市(CITY)组成;

供应情况表 SPJ 由供应商代码(SNO)、零件代码(PNO)、工程项目代码(JNO)、供应数量(QTY)组成,表示某供应商供应某种零件给某工程项目的数量为 QTY。

(1) 用 SQL 语句建立以上四个表。

(2) 找出所有供应商的姓名和所在城市。

(3) 找出所有零件的名称、颜色、重量。

(4) 找出使用供应商 $S1$ 所供应零件的工程号码。

(5) 找出工程项目 $J2$ 使用的各种零件的名称及其数量。

(6) 找出上海厂商供应的所有零件号码。

(7) 找出使用上海产的零件的工程名称。

(8) 找出没有使用天津产的零件的工程号码。

(9) 把全部红色零件的颜色改成蓝色。

(10) 由 $S5$ 供给 $J4$ 零件 $P6$ 改为由 $S3$ 供应,请做必要的修改。

(11) 从供应商关系中删除 $S2$ 的记录,并从供应情况关系中删除相应的记录。

(12) 请将($S2$,$J6$,$P4$,200)插入供应情况关系。

第4章 关系数据理论

这一章讨论关系数据理论。从数据存储异常问题引出关系数据库的规范化设计理论。规范化设计理论对关系数据库结构的设计起着重要作用。本章介绍最重要的一种数据依赖——函数依赖，关系模式进一步规范化的内容，以及关系模式的分解。

4.1 数据存储异常

数据库系统设计的关键是数据模式的设计，即如何把现实世界表达成一个合适的数据模式。由于关系模式有严格的数学理论基础，并且关系模式不但可以用二维表描述实体，还可以用二维表描述实体之间的联系，因此，人们通常以关系模式为背景来讨论这个问题，就形成了数据库逻辑设计的一个有力工具：关系数据库的规范化理论。

4.1.1 关系模式设计概述

数据库设计的一个最基本的问题是如何建立一个好的数据库模式，即给出一组数据，如何构造一个适合于它们的数据模式，使数据库系统无论是在数据存储方面，还是在数据操纵方面都有较好的性能。在第1章讨论数据模型时，指出了E-R模型常被作为描述现实世界的数据模型。除了面向对象的数据库设计，E-R方法的确是当前设计数据库的基本方法。人们总是先从现实世界得到E-R模型，然后，再将E-R模型转换成各类数据库系统支持的数据库模式。最后一步工作可以用第7章介绍的“机械”的方法完成。无论通过E-R模型的方法，还是别的方法得到的数据库设计，都可能有数据冗余，而冗余可能会引起其他不希望出现的异常，不能保证设计的数据库具有好的性能。

在E-R模型提出以前，Codd提出了关系数据库理论，并发展了一套关系数据库设计的理论，即关系的规范化理论。这套理论根据现实世界存在的数据依赖进行关系模式的规范化处理，从而得到一个好的数据库设计。数据依赖实际上是一种关系上的语义完整性约束，是对关系模式任何时刻的实例取值的限制。如果定义了关系的关键字，也就定义了“该关系的所有属性依赖于主关键字属性(主属性)”的依赖。如果定义了外部关键字，当将这个关系和外部关键字指引的关系合并(自然连接)时，新生成的关系中从被指引的关系得到的所有属性依赖于外部关键字(它在新关系中不一定是关键字)。规范化后的关系模式至少可以避免许多不希望出现的异常。当然，找出所有的数据依赖并不是一件容易的事，而且纯粹根据存在的数据依赖进行关系规范化，所得到的数据库设计(在实际中)也不一定是最优的，因为

此时没有考虑关系的实际大小和对关系经常进行什么样的操作。

但是,关系数据库设计的理论仍有实用价值。首先,关系数据库设计的理论能帮助用户分析和判别什么是一个好的数据库模式,甚至判别怎样的 E-R 模型可转化出好的数据库模式;其次,从 E-R 模型转化得到关系模式后可再用关系规范化优化;第三,正如第 7 章指出的,用机械的方法从 E-R 模型转化得到的关系模式有时很烦琐,关系数据库设计的理论可以指导我们合并关系模式,以精简设计。所以,流行的关系数据库设计方法是先得到 E-R 模型,然后转化为关系模式,再进行关系模式的优化。

4.1.2 关系模式的数学表示

关系模式是对关系的描述,可以将关系模式形式定义成一个五元组:

$$R(U, D, \mathrm{DOM}, F)$$

其中:

(1) R 为关系名;

(2) U 为一组属性,即组成 R 的全部属性的集合;

(3) D 为域的集合,即属性取值范围的集合;

(4) DOM 为 U 与 D 之间的映像;

(5) F 为属性组 U 上的数据依赖关系的集合。

关系是关系模式在某一时刻的状态或内容。关系模式是静态的、稳定的,关系是动态的,不同时刻关系模式中的关系可能会不同。但是,每一个关系都要满足关系模式的数据依赖关系集合 F 的约束条件。

由于在关系模式 $R(U, D, \mathrm{DOM}, F)$ 中,影响数据库设计的主要因素是 U 和 F,D 和 DOM 对设计的影响不大,可以忽略。这样,关系模式可以定义为三元组 $R(U, F)$ 当且仅当 U 上的一个关系 r 满足 F 时,r 称为关系模式 $R(U, F)$ 的一个关系。

4.1.3 实例分析

在数据库中怎样才能建立一个好的关系数据库模式呢?在解决如何设计一个好的关系数据库模式之前,首先看一看什么是坏的数据库设计。

【例 4.1】 教师授课管理系统。在实际当中进行教师授课的人工管理,通常是制作一张表(如表 4-1 所示)。

表 4-1 教师授课管理系统使用的二维表

教师号	教师姓名	联系电话	课程号	课程名
t01	李桦	5523227	j01	计算机文化基础
			j02	微机原理
t02	王伟	5580923	j01	计算机文化基础
			j03	C 语言程序设计
			j04	计算机软件基础
t03	李琳	4756859	j03	C 语言程序设计

管理人员通过查询表 4-1 来了解某个教师可以教授哪些课程，以便安排教学任务、通知授课教师。对这个实际问题，把它设计成一个教师授课管理的计算机系统，首先要将表 4-1 中的数据描述成关系型数据库的关系数据模型，然后将表格数据存储到计算机中。关系模型是一个标准的二维表，要准确存储表 4-1 中的数据，可以将表 4-1 转换为表 4-2 所示的表格。

表 4-2　教师授课管理系统改进后的二维表

教师号	教师姓名	联系电话	课程号	课程名
t01	张明	552322	j01	计算机文化基础
t01	张明	552322	j02	微机原理
t02	李华	558092	j01	计算机文化基础
t02	李华	558092	j03	C 语言程序设计
t02	李华	558092	j04	计算机软件基础
t03	王伟	475685	j03	C 语言程序设计

可以看到，表 4-2 完全描述了表 4-1 所描述的信息，并且表 4-2 是一个关系，即二维表，对应的关系模式为：R(教师号，教师姓名，联系电话，课程号，课程名)。

由现实世界中的事实可知：一个教师只有一个教师姓名、电话；一门课程只有一个课程号。

于是关系模式 R 的关键字是(教师号，课程号)，即根据每个教师号和所教课程的编号就能确定课程名称，根据每个教师号就能确定教师姓名、联系电话。虽然这个模式只有五个属性，但在使用过程中明显存在下列问题。

1. 数据冗余

在上述关系模式中，如果一名教师可以教授多门课程，那么这名教师的教师号、教师姓名、联系电话就要重复多次，如教师张明、李华。

2. 更新异常

由于数据的冗余，在数据更新时会出现问题。例如，一个教师教三门课程，在关系中就会有三条记录。如果这个教师的地址改变了，这三条记录中的联系电话都要改变；若在修改的过程中，有一条记录的联系忘记更改了，就会造成这个教师联系电话不一致的错误。

3. 插入异常

在关系型数据库中，关键字是能够唯一标识记录的属性或属性组，并且在关系中，作为关键字的属性不能为空值。也就是说，系统规定：在关系中，当作为关键字的属性没有值时，这样的记录是非法记录。而实际情况是，如果新增加一名教师，尚未分配教学任务(没有课程号)，那么教师的教师号、教师姓名和联系电话就无法形成记录，存储到关系中去，因为课程号是标识课程的关键字，不能为空。同样，对于一门新课程，如果没有安排任课教师，则课程信息也无法存储到关系中去。

4. 删除异常

与插入问题相反，删除操作会造成一些信息的丢失。例如，一个教师原来有教学任务，

目前没有安排教学任务，如果把这个教师的所有记录都删除，就把这个教师的姓名和联系电话的信息也从数据库中删去了，以后要给这个教师安排教学任务，则无处查询其相关信息，这也是一种不合理的现象。更为严重的是，如果删除了一些教师信息，可能会无意间将这些教师所授课程信息也一并删除了；或者因为删除某些课程信息，而错误地删除了一些教师信息。

如果将上述模式分解成下面三个关系模式：*R*1(教师号，教师姓名，联系电话)；*R*2(课程号，课程名)；*R*3(教师号，课程号)。

对应的关系实例如表4-3、表4-4和表4-5所示。

表4-3 *R*1

教师号	教师姓名	联系电话
t01	张明	552322
t02	李华	558092
t03	王伟	475685

表4-4 *R*2

课程号	课程名	课程号	课程名
j01	计算机文化基础	j03	C语言程序设计
j02	微机原理	j04	计算机软件基础

表4-5 *R*3

教师号	课程号	教师号	课程号
t01	j01	t02	j03
t01	j02	t02	j04
t02	j01	t03	j03

可以看到表4-3、表4-4和表4-5完全描述了表4-2的所有信息，并且对表4-2进行了分解：表4-3描述教师的基本信息；表4-4描述课程的基本信息；表4-5描述教师讲授课程的信息。这样就解决了前面提到的数据冗余、更新异常、插入异常和删除异常的问题，即每个教师的信息(教师号、教师姓名、联系电话)只存放一次，不管这个教师有没有教学任务，他的信息均在关系*R*1中存放，数据的更新、插入和删除均不会产生上述操作异常。

分析关系模式*R*，其主关键字为属性组(教师号，课程号)，非主属性中教师名、联系电话由教师号唯一确定；非主属性中课程名由课程号唯一确定。分析*R*1、*R*2、*R*3，*R*1的主关键字是教师号，非主属性教师名、联系电话严格依赖于教师号；*R*2的主关键字是课程号，非主属性课程名由课程号唯一确定；*R*3的主关键字为属性组(教师号，课程号)，无非主属性。可以这么说，关系模式是否合适与关系中非主属性对主属性的数据依赖有关。

当然，对分解了的关系进行一些复杂的查询操作时，就必须进行关系的连接运算，这增大了查询运算的代价；而在原来的单个关系中，则只需要进行单个关系上的选择和投影运算。那么，如何确定关系的分解是否有益？分解后是否仍然存在数据冗余和更新异常等问题？怎样的关系模式才好？这将是本章所要讨论的问题。

在这里，需要强调一下数据依赖和冗余之间的关系。一般说来，依赖是对关系模式任何

时刻的实例的值的约束，只有取满足约束条件的值构成的关系才是合法的关系，它表示了现实世界的某一可能状态。同时，根据数据间的依赖关系，可以由一些数据推导出另一些数据。比如，知道了“t01”教师的姓名和联系电话，又知道了“j01”课程和“j02”课程由同一个教师讲授，就可以推导出“j02”课程对应的教师姓名和联系电话。因此，我们可以利用依赖关系避免存储冗余的数据，同时也避免了冗余带来的其他异常。然而，并不是一切冗余都不好，在合法的关系中有某些冗余也是合乎情理的，但必须采取措施避免冗余带来的数据不一致。

如果依赖是函数依赖的形式，则冗余的形式是明显的。例如，在关系 R(供应商，地址，产品，价格)中看到有两个元组 $t1$(“王刚”，“解放路 18 号”，“彩电”，3000)和 $t2$(“王刚”，“???”，“DVD”，1000)，则可以假设供应商能决定地址，即地址依赖于供应商，从而推导出“???”代表“解放路 18 号”。因此，函数依赖使得对一给定的供应商，除第一个元组地址字段值外，其余地址字段的值都是冗余的，不用查看就能知道。反过来，如果地址和供应商之间不存在函数依赖，即一个供应商有多个地址，则不能推导出“???”的值，这个字段也就不是冗余的了。

当存在比函数依赖更一般的依赖时，冗余存在形式就不明显了。但是在所有的情况下，消除冗余的办法和冗余的导致因素有关。例如，要消除函数依赖引起的冗余，可以根据函数依赖关系来实现。

4.2 函数依赖

数据依赖是指通过一个关系中属性之间值的相等与否体现出来的数据间的相互关系，是现实世界事物之间相互关联性的一种表达，是属性固有语义的体现。大多数数据依赖都是函数依赖。

4.2.1 函数依赖的一般概念

关系模式的定义实际上只需要指出关系的目数，但在很多情况下，即使某些元组有正确的目数，而且各分量取自正确的域中，也不一定就是某个关系的元组。一般，在关系上还应有两类限制：

- 元组分量的语义限制。例如，没有 5 米高的人，也没有年龄 27 岁而工龄 35 年的职员。这些限制可以采用 DBMS 来进行检查，这些不合情理的值可能是由于错误的输入或计算造成的。因此，在设计数据库模式时，必须进行数据完整性的约束。
- 关系中属性间值相等或不等的限制。有一些约束不取决于元组中分量的取值范围，而是要求两个元组在确定的分量上保持一致，这类约束中最重要的是函数依赖。

为了方便起见，假设 $R(A_1, A_2, A_3, \cdots, A_n)$是一个关系模式，$U=\{A_1, A_2, A_3, \cdots, A_n\}$是 R 的所有属性的集合，X、Y 和 Z 表示 R 中的属性子集。

1. 函数依赖

定义 4.1 设 $R(U)$是属性集 U 上的关系模式，X 和 Y 是 U 的子集。若对于 $R(U)$的任

意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等，而在 Y 上的属性值不等，则称 X 函数决定 Y 或 Y 函数依赖于 X，记作 $X \to Y$，X 称为决定因素。

一个函数依赖成立就意味着关系在任何一刻的实例都满足函数依赖条件。要确定一个函数依赖关系，需要弄清楚数据的语义，而数据的语义是现实世界的反映，不是主观的臆断或由关系的当前实例所决定的。例如在百货商店的顾客关系中，若每个顾客都只有一个地址，且没有重名的顾客，则顾客地址对顾客姓名的函数依赖可表示成：Cname→Adrr。

因此需要注意：

(1) 函数依赖 $X \to Y$ 的定义，强调模式 R 的任意具体关系 r 应具有的特性，而不是某个或某几个具体关系具有的特性。

(2) 函数依赖 $X \to Y$ 的定义，强调具体关系的任意两条记录具有的特性，而不是某两条记录具有的特性。

(3) 函数依赖关系是自然产生的，如教师姓名函数依赖于教师号，教师的联系电话函数依赖于教师号。

下面介绍一些术语与记号：

- $X \to Y$，但 Y 不是 X 的子集，则称 $X \to Y$ 是非平凡函数依赖。
- $X \to Y$，但 Y 是 X 的子集，则称 $X \to Y$ 是平凡函数依赖。
- 若 $X \to Y$，则 X 称为这个函数依赖的决定因素，或决定属性集。
- 若 $X \to Y$，$Y \to X$，则记作 $X \leftarrow\rightarrow Y$。
- 若 Y 不函数依赖于 X，则记作 $X \nrightarrow Y$。

定义 4.2 在关系模式 $R(U)$ 中，对于 U 的子集 X 和 Y，如果 $X \to Y$，但 $Y \not\subset X$，则称 $X \to Y$ 是非平凡函数依赖；如果 $Y \subseteq X$，则称 $X \to Y$ 是平凡函数依赖。

显然，对于任一关系模式，平凡函数依赖都是成立的，因此，如果不特别声明，我们讨论的都是非平凡函数依赖。

定义 4.3 在关系模式 $R(U)$ 中，如果 $X \to Y$，并且对于 X 的任何一个真子集 X'，都有 $X' \nrightarrow Y$，则称 Y 对 X 完全函数依赖，记作 $X \xrightarrow{f} Y$。

对于如下关系：教室(课程，班级，时间，教室)，其中存在{课程，班级，时间}$\xrightarrow{f}${教室}，因为一间教室在不同的时间可供多个班级上多门课程，只有课程、班级和时间都确定以后，教室才唯一确定。

若 $X \to Y$，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖，记作 $X \xrightarrow{p} Y$。

例如，教师姓名→教师号为完全函数依赖，(教师号，课程号)→课程名为部分函数依赖。

定义 4.4 在关系模式 $R(U)$ 中，如果 $X \to Y$，$Y \to Z$，$Y \nrightarrow X$，且 Y 不是 X 的子集，Z 不是 Y 的子集(均为非平凡函数依赖)，则称 Z 传递函数依赖于 X，记作 $X \xrightarrow{t} Z$。

加上条件 $Y \nrightarrow X$，是因为如果 $Y \to X$，则 $X \leftarrow\rightarrow Y$，实际上是 $X \to Z$，是直接函数依赖，而不是传递函数依赖。

例如，关系 R(<u>SNO，CNO，</u>SDEPT，SLOC，CNAME)，其中 SNO 为学生的学号，CNO 为课程号，SDEPT 为学生所在系，SLOC 为学生的住处(每个系的学生住在同一个地方)，CNAME 为课程名称。在关系 R 中，SNO→SDEPT，而 SDEPT→SLOC，SNO→SLOC 为传

递函数依赖。

例如,对如下关系:教师(姓名,职务,职务工资),其中,姓名→职务,职务→职务工资,则姓名→职务工资,所以,姓名$\xrightarrow{t}$工资。

定义 4.5 设 F 是 R 的函数依赖集合,$X \to Y$ 是 R 中的一个函数依赖,如果对于 R 中任何一个满足 F 的取值,都必然满足 $X \to Y$,则称 F 逻辑蕴涵 $X \to Y$,或称 $X \to Y$ 可以由 F 推导。被 F 所逻辑蕴涵的函数依赖的全体称为 F 的闭包(Closure),记为 F^+。一般情况下 $F \subseteq F^+$,当 $F = F^+$ 时,称 F 是一个函数依赖的满族。

函数依赖是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系,它决定于数据的语义,而与具体的关系无关。函数依赖是模式级的概念,是数据模型的组成部分。定义关系模式上属性组之间存在的函数依赖关系的条件是这个函数依赖必须在关系模式的所有实例上成立,而不是在个别实例上成立。也就是说,不能从具体的关系模式的实例判断某属性组函数依赖于某属性组,只能判断一个关系模式的实例满足某函数依赖。例如,在 4.1 节给出了"学生"关系模式的一个实例,考察这个关系中已有的元组,可以断定该实例满足函数依赖:系名→系主任,但不能由此得出"学生"关系模式中存在函数依赖:系名→系主任,因为所考察的只是一组有限的局部数据,不能排除可能有一些特殊的系有两个并列的系主任。函数依赖只能根据现实世界的语义确定,从这种意义上讲,函数依赖的确定实际上是对现实世界的论断。函数依赖一旦确定,任何时候(过去、现在和将来)关系的所有实例都应满足这个函数依赖。由于关系将来具体的取值不能一一列出并进行验证,所以函数依赖只能在对现实世界进行调查和分析以后,根据数据客观存在的联系和企业或组织管理的规章制度来确定。

2. 关键字(码)

在第 2 章中已经提到,对于现实世界的每一实体集合,都有一关键字(本章称其为码),即唯一确定各个实体的一组属性。实体集合关键字的值唯一确定一个实体,从而确定该实体各属性的值。同样,关系上最重要的约束是关系的关键字约束,就是说关键字唯一确定关系的元组,从而确定元组各属性的值。在函数依赖的基础上,可以更确切地定义关键字概念。

定义 4.6 设 K 为关系模式 $R<U, F>$ 中的属性或属性组合,若 $K \xrightarrow{f} U$,则 K 为 R 的关键字,也称候选码。若关键字多于一个,则选定其中一个为主码。

包含在任何一个候选码中的属性,叫做主属性。不包含在任何码中的属性称为非主属性或非码属性。最简单的情况,单个属性是码。最极端的情况,整个属性组是码,称为全码。值得注意的是,K 一定是能唯一确定 R 中某一元组的最小属性集合。

【例 4.2】 关系模式 R(城市 Z,街道 S,邮编 Z),这一关系表明只有当城市 c 有一栋大楼的街道地址是 s,且该城市该地址的邮编是 z 时,R 的关系中才有元组(c, s, z),假定非平凡函数依赖有:{城市,街道}→{邮编},{邮编}→{城市},即地址决定邮编,而邮编尽管不决定街道地址,但决定城市。可以看出,其中{城市,街道}和{街道,邮编}都是关键字。

定义 4.7 关系模式 R 中属性或属性组合 X 并非 R 的码,但 X 是另一个关系模式的码,则称 X 是 R 的外部码,也称外码。

在关系模式学生选课(学号,课号,成绩)中,学号不是关键字,但学号是关系模式学生

(学号,姓名,年龄)的关键字,则学号对关系模式学生选课来说是外关键字。主关键字和外关键字提供了一个表示关系间联系的手段。

4.2.2 Armstrong 公理系统

为了从已知的函数依赖导出更多新的函数依赖,比如从已知的函数依赖集 F 推导出 F 的闭包 F^+ 的全部函数依赖,需要一套形式化的推理规则。1974 年,由 W. W. Armstrong 归纳整理了一套推理规则,即 Armstrong 公理系统。它是关系模式分解算法的理论基础。

设关系模式 $R(U, F)$,U 为 R 的属性集合,F 为 U 上的一组函数依赖,则对于 $R(U, F)$ 来说有以下推理规则:

(1) 自反律(Reflexivity):如果 $Y \subseteq X \subseteq U$,则 F 逻辑蕴涵 $X \rightarrow Y$。

(2) 增广律(Augmentation):如果 $X \rightarrow Y$ 为 F 所蕴涵,且,则 $XZ \rightarrow YZ$ 为 F 所蕴涵。

(3) 传递律(Transitivity):如果 $X \rightarrow Y$ 和 $Y \rightarrow Z$ 为 F 所蕴涵,则 $X \rightarrow Z$ 为 F 所蕴涵。

由关系的性质,可以证明 Armstrong 公理系统是有效的和完备的,在此从略。另外,根据上述的 Armstrong 公理可推导出下面三条推理规则:

(4) 合并规则(The-Uniori Rule):如果 $X \rightarrow Y$ 和 $X \rightarrow Z$ 成立,则 $X \rightarrow YZ$ 成立。

(5) 伪传递规则(The Pseudo Transitivity Rule):如果 $X \rightarrow Y$ 和 $WY \rightarrow Z$ 成立,则 $XW \rightarrow Z$ 成立。

(6) 分解规则(The Decomposition Rule):如果 $X \rightarrow Y$ 和 $Z \subseteq Y$ 成立,则 $X \rightarrow Z$ 成立。

由 Armstrong 公理系统,可以得到以下重要的结论:

- $X \rightarrow A_1, A_2, A_3, \cdots, A_n$ 成立的充分必要条件是 $X \rightarrow A_i$ 成立,其中 $i=1,2,\cdots,k$。
- 函数依赖集 F 的闭包 F^+ 是从 F 出发用公理导出的所有函数依赖的集合。

【例 4.3】 对 4.2 节的"学生"关系模式,假设通过调查某学校的管理模式和工作流程,略去一些次要的、不用数据库管理的部分信息,可以确定出如下非平凡的(完全)函数依赖,并对一些函数依赖的语义进行了解释:

学号→姓名,每个学生有唯一的一个学号;

系→系主任,每个系最多只有一个系主任;

学号→系,每个学生只能属于一个系;

(学号,课程)→成绩,一个学生选修一门课程有一个最终的成绩。

从以上函数依赖集合出发,利用 Armstrong 公理系统,可计算出该函数依赖集合的闭包,下面列出其中的一部分:

学号→系主任;

(学号,系)→系主任;

(学号,姓名)→系主任;

(学号,课程)→系主任;

(学号,课程)→系;

(学号,课程)→姓名;

…

从后面三个函数依赖可看出,(学号,课程)是关键字。如果学校的管理规章制度不同,

可能会有不同的函数依赖存在。例如，如果该校只允许学生选修本系提供的课程，则存在(系，课程)→系主任等函数依赖。

4.3 关系模式的规范化

在关系数据模式设计中，为了避免由依赖引起的数据的冗余和更新异常问题，必须进行关系数据模式的合理分解，即进行关系数据模式的规范化。自1971年E. F. Codd提出关系规范化理论开始，人们对数据库模式的规范化问题进行了长期的研究，并已经有了很大进展。

关系数据库中的关系是要满足一定规范化要求的，对于不同的规范化程度，可以用“范式”来衡量，记为xNF。范式是表示关系模式的级别，是衡量关系模式规范化程度的标准，达到范式的关系才是规范化的。满足最低要求的为第一范式，简称1NF。在第一范式的基础上，进一步满足一些要求的为第二范式，简称2NF，……，依此类推，各种范式之间的联系是5NF⊂4NF⊂BCNF⊂3NF⊂2NF⊂1NF。

范式的概念最早是由E. F. Codd提出的，1971—1972年，他先后提出了1NF、2NF、3NF的概念，1974年后，他又和Boyce共同提出了BCNF的概念，1976年，Fagin提出了4NF的概念，后来又有人提出了5NF的概念。在这些范式中，最重要的是3NF和BCNF，这是进行规范化的主要目标。

一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化。

4.3.1 第一范式

定义4.8 如果一个关系模式R的所有属性都是不可分的基本数据项，则$R \in$1NF。

在第一范式中只要求关系模式的关系是标准的二维表，没有论及关系模式中所存在的函数依赖关系，这种范式是关系模式最基本的要求。

这是关系模式必须达到的最低要求，不满足该条件的关系模式称为非规范化关系，即非第一范式。目前大部分商用的RDBMS(关系数据库管理系统)处理的关系要求至少是1NF的，但一些新型的RDBMS，比如ORDBMS(对象-关系数据库管理系统)支持非1NF。

关系模式R(<u>SNO，CNO，</u>SDEPT，SLOC，CNAME，GRADE)，其函数依赖如图4-1所示。

图4-1中用虚线表示部分函数依赖。可以看到非主属性SDEPT、SLOC并不完全函数依赖于码(<u>SNO，CNO</u>)。这样，关系R存在以下几个问题：

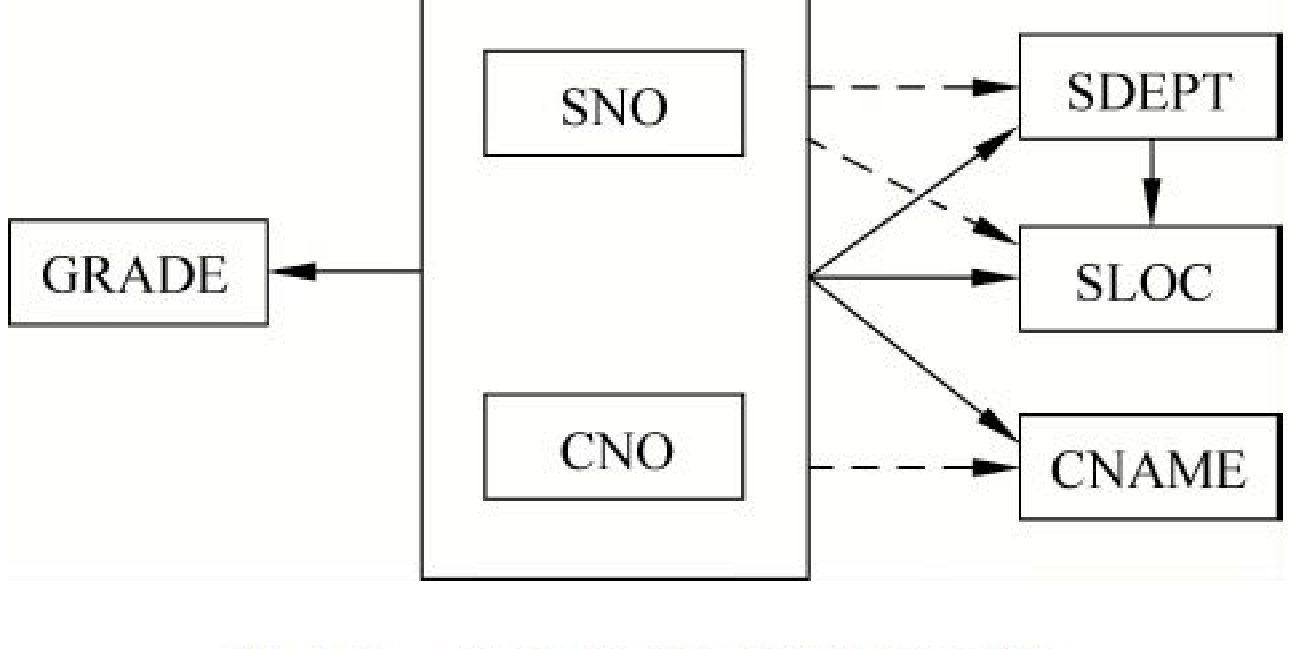

图4-1　关系R的函数依赖图示

(1) 数据冗余。如果某系有 1000 个学生,则这 1000 学生记录的 SLOC 数据项将重复 1000 次。除此之外,只要某门课被不同学生选择,其 CNAME 数据项将也要重复。可见,此关系数据冗余量太大。

(2) 插入异常。假若要插入一个学生,但该生还未选课,即这个学生无 CNO,这样的元组就插不进 R 中。因为插入元组时必须给定码值,而这时码值的一部分为空,因而学生的固有信息无法插入。

(3) 删除异常。假定某个学生开始只选了一门课,后来连这门课也不选了,那么此记录的 CNO 数据项就会删除,可是 CNO 为主属性,删除了该数据项,整个元组就必须跟着删除,使得此学生的其他信息也被删除了,从而造成删除异常,即不应删除的信息也被删除了。

(4) 修改复杂。某个学生从数学系(MA)转到计算机科学系(CS),这本来只需修改此学生元组中的 SDEPT 分量。但因为关系模式 R 中还含有系的住处 SLOC 属性,学生转系将同时改变住处,因而还必须修改元组中的 SLOC 分量。另外,如果这个学生选修了 K 门课,SDEPT、SLOC 重复存储了 K 次,则必须无遗漏地修改 K 个元组中全部 SDEPT、SLOC 信息,造成修改的复杂化。

4.3.2 第二范式

定义 4.9 若关系模式 $R\in 1NF$,且每一个非主属性都完全函数依赖于码,则 $R\in 2NF$。

从上面的例子可以发现,问题在于非主属性 SDEPT、SLOC、CNAME 对码不是完全函数依赖。可以用投影分解把关系模式 R 分解为两个关系模式:SC(SNO,CNO, GRAGE)、C(CNO,CNAME)和 S(SNO, SDEPT,SLOC),这样,关系模式就成为了 2NF。

例如,有以下关系模式,表示零件的库存:库存(零件号,仓库号,零件数量)。

假设同一种零件可能存放在多个仓库中,同一仓库可存放不同种类的零件,则仅有一个非平凡的函数依赖存在:(零件号,仓库号)→零件数量。此关系是 1NF 的,且非主属性零件数量是完全函数依赖于码{零件号,仓库号}的,因为若只确定零件号或仓库号,零件数量都是不能确定的,所以库存关系是 2NF 的。

关系模式"学生(学号,姓名,系名,系主任,课程号,成绩)"是 1NF 的,其码为{学号,课程号},而非主属性中,只有成绩是完全函数依赖于码,姓名、系名、系主任对码都是部分函数依赖,因为只需要学号就可以确定它们的值,所以学生关系模式不是 2NF 的。

可以用简单的投影分解来使非 2NF 的关系转变为 2NF 的关系。将部分函数依赖关系中的主属性(决定方)和非主属性从关系模式中提出,单独构成一个关系模式;将关系模式中余下的属性,加上码(仍要保留部分函数依赖的决定方属性,起分解出来的新关系之间的关联作用)构成另一关系模式。例如,"学生"关系模式就可分解成两个 2NF 的关系模式:学生记录(学号,姓名,系,系主任);成绩(学号,课程,成绩)。

这里说投影分解,意思是分解所得关系的实例是原关系实例的投影:

$$r(\text{学生记录}) = \pi_{(\text{学号},\text{姓名},\text{系名},\text{系主任})}(r(\text{学生}))$$

$$r(\text{成绩}) = \pi_{(\text{学号},\text{课程},\text{成绩})}(r(\text{学生}))$$

分解后,4.2 节中给出的四个函数依赖仍然成立:

学号→姓名,每个学生有唯一的一个学号;

系→系主任,每个系最多只有一个系主任;

学号→系，每个学生只能属于一个系；

(学号，课程)→成绩，一个学生选修一门课程有一个最终的成绩。

4.3.3　第三范式

但是将一个1NF关系分解为多个2NF的关系，并不能完全消除关系模式中的各种异常情况和数据冗余。

例如，2NF关系模式 S(SNO，SDEPT，SLOC)中有下述函数依赖：SNO→SDEPT，SDEPT→SLOC，SNO→SLOC，也就是说 S 中存在非主属性对码的传递函数依赖。关系 S 中仍然存在插入异常、删除异常、数据冗余和修改复杂的问题。

定义 4.10　关系模式 $R(U,F)$中若不存在候选码 X、属性组 Y 及非主属性 $Z(Z\not\subset Y)$，使得 $X\rightarrow Y(Y\nrightarrow X)$，$Y\rightarrow Z$ 成立，则 $R(U,F)\in$ 3NF。

若 $R\in$ 3NF，则每一个非主属性既不部分函数依赖于码也不传递函数依赖于码。

关系模式 S 出现上述问题的原因是存在非主属性 SLOC 对码 SNO 的传递函数依赖。解决办法是对 S 投影分解，得到 SD(SNO，SDEPT)和 DL(SDEPT，SLOC)。

4.3.4　BC 范式

属于3NF的关系模式可以在一定程度上解决原2NF关系中存在的插入异常、删除异常、数据冗余度大、修改复杂等问题，但并不能使其完全消除。

例如，关系模式 ORDER(C,F,P)中，C 表示用户，F 表示厂家，P 表示产品。假设某用户使用某产品从一个厂家进货，某产品可由多个厂家生产，但一个厂家只生产一种产品。则关系 ORDER 中(C,F)和(C,P)均为候选码，因为没有任何非主属性部分函数依赖和传递依赖于码，因此 ORDER∈3NF。但是该模式仍存在以下问题：

(1) 数据冗余度大：虽然某厂家只生产一种产品，可是选购该厂家产品的用户都要记录此产品信息。

(2) 修改操作复杂：当某个厂家的产品改变名称后，所有选择该厂家产品的用户都要修改此信息。

(3) 增加操作异常：当生产某产品的厂家增加了一个新的，可是因为还没有用户选择他，则此厂家生产该产品的信息将无法加入。

(4) 删除操作异常：某厂家的所有用户均不选择该厂家的产品时，就需要删除这些记录，可是该厂家生产的产品信息也就不存在了，这是不应该的。

之所以这样，是因为关系 ORDER(C,F,P)中的存在以下函数依赖：$(C,P)\rightarrow F$，$(C,F)\rightarrow P$，$F\rightarrow P$。也就是说，主属性 P 部分函数依赖于码(C,F)，需要进一步规范化为 BCNF。

定义 4.11　关系模式 $R(U,F)\in$ 1NF。若对于 R 的每一个函数依赖 $X\rightarrow Y$，且 $Y\not\subset X$，则 X 必含有码，那么 $R(U,F)\in$ BCNF，即 R 中每一个决定因素都包含码，则 R 是 BCNF。

上述关系 ORDER(C,F,P)可以分解为 $CF(\underline{C,F})$和 $PF(\underline{F},P)$，这样在关系模式中消除了主属性对码的部分函数依赖，都成为了 BCNF。

每个 BCNF 的关系模式都具有以下性质：

- 所有非主属性都完全函数依赖于每个候选码。

- 所有的主属性也完全函数依赖于每一个不包含它的候选码。
- 没有任何属性完全函数依赖于非码的任何一组属性。

属于 3NF 的关系模式有的属于 BCNF，但有的不属于 BCNF。假设 R 只有一个候选码，那么，若 $R \in$ 3NF，则 R 必属于 BCNF。

由 BCNF 的定义可以知道，一个满足 BCNF 的关系模式一定是非主属性对码完全函数依赖；主属性对不包含它的码也是完全函数依赖；没有属性完全函数依赖于码以外的属性组。下面是一个 BCNF 的关系：学生(学号，姓名，专业，宿舍)；该关系中，假定学生无重名的，则码为{学号}或{姓名}，非主属性对这两个码不存在部分函数依赖和传递函数依赖，因此是 3NF 的，而同时除{学号}和{姓名}以外没有其他决定因素，故该关系也是 BCNF 的。再看一个不是 BCNF 的例子：

通讯(城市，街道，邮政编码)；

在该关系中，码为{城市，街道}，非主属性{邮政编码}完全函数依赖于码，且无传递依赖，属于 3NF，但如果邮政编码确定了，城市也就确定了，它是一个决定因素，而这个决定因素不是码，也不包含在码中，所以该关系不是 BCNF 的。

值得注意的是 3NF 与 BCNF 之间的关系，一个关系模式属于 BCNF 则一定属于 3NF，BCNF 是 3NF 的特例，但反之则不然，属于 3NF 的关系不一定是 BCNF，3NF 是 BCNF 放宽一个限制，即允许决定因素不包含码。

提出 BCNF 的目的是消除函数依赖所产生的冗余，在 BCNF 关系中，没有一个属性值可只用函数依赖从任何其他属性值中推测出来。

4.3.5 多值依赖和第四范式

1. 多值依赖

定义 4.12 设 $R(U)$ 是属性集 U 上的一个关系模式。X、Y、Z 是 U 的子集，并且 $Z=U-X-Y$，多值依赖 $X \to\to Y$ 成立，当且仅当对 R 的任一关系 r，r 在 (X,Z) 上的每个值对应一组 Y 的值，这组值仅仅决定于 X 值而与 Z 值无关。

若 $X \to\to Y$，而 $Z=\Phi$，则称 $X \to\to Y$ 为平凡的多值依赖，否则称 $X \to\to Y$ 为非平凡的多值依赖。

多值依赖性质：

- 互补性：若 $X \to\to Y$，则 $X \to\to U-X-Y$；
- 对称性：若 $X \to\to Y$，$Z=U-X-Y$，则 $X \to\to Z$；
- 传递性：若 $X \to\to Y$，$Y \to\to Z$，则 $X \to\to Z-Y$；
- 增广性：若 $X \to\to Y$，$V \subseteq W$，则 $WX \to\to VY$；
- 若 $X \to Y$，则 $X \to\to Y$；
- 若 $X \to\to Y$，$Z \subseteq Y$，$Y \cap W=\Phi$，$W \to\to Z$，则 $X \to\to Z$；
- 若 $X \to\to Y$，$X \to\to Z$，则 $X \to\to YZ$；
- 若 $X \to\to Y$，$X \to\to Z$，则 $X \to\to Y \cap Z$；
- 若 $X \to\to Y$，$XY \to\to Z$，则 $X \to Z-Y$。

例如,关系模式 STORE(W,S,C)中,W 表示仓库,S 表示保管员,C 表示产品。假设某工厂有若干个仓库,生产若干种产品,每个仓库有若干保管员,有不同的产品,每个保管员保管所在的仓库的所有产品。列出关系如表 4-6 所示。

表 4-6　关系模式 STORE 的关系集合

W	S	P
$W1$	$S1$	$P1$
$W1$	$S1$	$P2$
$W1$	$S1$	$P3$
$W1$	$S2$	$P1$
$W1$	$S2$	$P2$
$W1$	$S2$	$P3$
$W2$	$S3$	$P2$
$W2$	$S3$	$P4$
$W2$	$S3$	$P5$
…	…	…

关系模式 STORE(W,S,P)具有唯一的候选码($\underline{W,S,P}$),即全码,所以 STORE∈BCNF,但仍存在以下问题:

(1) 数据冗余度大:对于某仓库来说,无论谁是保管员,保管的产品是一样的,因此,有多少保管员,产品就要存储多少次。

(2) 更新操作复杂:当仓库 $W2$ 增加、删除一位保管员时,就需要插入或删除 3 条记录;当某仓库保管的某产品发生变化时,该仓库有多少保管员就需要修改多少条记录。

之所以出现这样的问题,是因为保管员和产品的取值是毫无关系的,实际上它们都只由仓库决定,即它们之间存在多值依赖。

2. 第四范式

定义 4.13　关系模式 $R(U,F)\in$ 1NF,如果对于 R 的每个非平凡多值依赖 $X\rightarrow\rightarrow Y(Y\not\subseteq X)$,$X$ 都含有码,则称 $R(U,F)\in$ 4NF。

4NF 就是限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。4NF 所允许的非平凡的多值依赖实际上是函数依赖。

在关系模式 STORE 中,$W\rightarrow\rightarrow S$,$W\rightarrow\rightarrow P$,它们都是非平凡的多值依赖。而 W 不是码,关系模式 STORE 的码是($\underline{W,S,P}$),即全码。因此 STORE∉4NF。

解决办法是进行投影分解,消去非平凡且非函数依赖的多值依赖。例如,把 STORE 分解为 $WS(W,S)$和 $WC(W,P)$。在 WS 中虽然有 $W\rightarrow\rightarrow S$,但这是平凡的多值依赖,$WS$ 中已不存在非平凡且非函数依赖的多值依赖,所以 $WS\in$ 4NF,同理 $WP\in$ 4NF。

函数依赖和多值依赖是两种最重要的数据依赖。如果只考虑函数依赖,则属于 BCNF 的关系模式规范化程度已经是最高的了;如果考虑多值依赖,则属于 4NF 的关系模式规范化程度是最高的。

4.3.6 多值依赖和第五范式

1. 连接依赖

在数据依赖中，除了函数依赖和多值依赖以外，还有一种连接依赖。但连接依赖不像函数依赖和多值依赖可由语义直接导出，而是在关系的连接运算中才能反映出来。存在连接依赖的关系模式同样会遇到数据冗余和更新异常的情况，如果在 4NF 的基础上进一步投影分解，消除连接依赖，就成为了 5NF。目前，5NF 是数据库的最高范式。

定义 4.14 设 $X_1, X_2, \cdots, X_n$ 是关系 R 的属性子集，且 $X_1 \cup X_2 \cup \cdots \cup X_n = U$。若对 R 的任一取值，$R = R(X_1) \bowtie R(X_2) \bowtie \cdots \bowtie R(X_n)$ 均成立，则称 R 具有连接依赖，记为 $\bowtie(X_1, X_2, \cdots, X_n)$。

如果 $X_i = U(i=1,2,\cdots,n)$，则连接依赖总是成立的，这称为平凡连接依赖，否则称为非平凡连接依赖。平凡连接依赖无实际意义。多值依赖可以看成是连接依赖的一个特例。一般的连接依赖不容易直观地从数据语义中发现，是一种普遍化程度较高的数据依赖，在数据库设计中，几乎不需要考虑这种数据依赖，因为它对数据库性能的影响不大，而且考虑到数据库的查询效率，一般允许这样的数据依赖存在。

2. 第五范式

如果关系模式 $R \in$ 4NF，且除了由超键（包含一个候选关键字的属性组）构成的连接依赖外，别无其他连接依赖，即 R 中每一个连接依赖均由 R 的候选码所隐含，则称关系模式 R 是 5NF，即 $R \in$ 5NF。

所谓 R 中每一个连接依赖均由 R 的候选码所隐含是指在连接时，其连接属性必须是 R 的候选码。下面来看一个不是 5NF 的关系模式：

提供（供应商号，零件号，工程号）；

其中，唯一的候选码是全码{供应商号，零件号，工程号}，且该关系模式可不丢失数据地投影分解成三个关系模式：供应（供应商号，零件号），需要（工程号，零件号）和承接（供应商号，工程号），而这三个关系模式不包含原关系的候选码，所以原关系不是 5NF 的，而且原关系模式也不是 4NF 的。

接下来再看一个属于 5NF 的关系模式：

学生（学号，姓名，性别，年龄，来自地区，入学成绩）；

显然该关系是 4NF 的，且其候选码只有一个{学号}，而关系模式满足的所有连接依赖的连接属性都一定是学号，所以该关系模式是 5NF 的。

任何关系模式总是可以无损失地投影分解成一组 5NF 的关系，同时又可以用连接来重构原关系模式，因此 5NF 相对于投影和连接运算来说已是最高范式。

4.3.7 规范化过程小结

规范化：把低一级的关系模式通过模式分解，转化为若干个高一级范式的关系模式的集合的过程。

目的：解决关系模式存在的插入、删除异常、修改复杂，数据冗余等问题。

基本思想：“分离”，“一事一地”。让一个关系描述一个概念、一个实体或者实体间的一种联系。若多于一个概念则把它“分离”出去。

实质：概念的单一化。

步骤：分解关系模式，把低一级的关系模式分解为若干个高一级的关系模式（如图 4-2 所示）。这种分解不唯一。

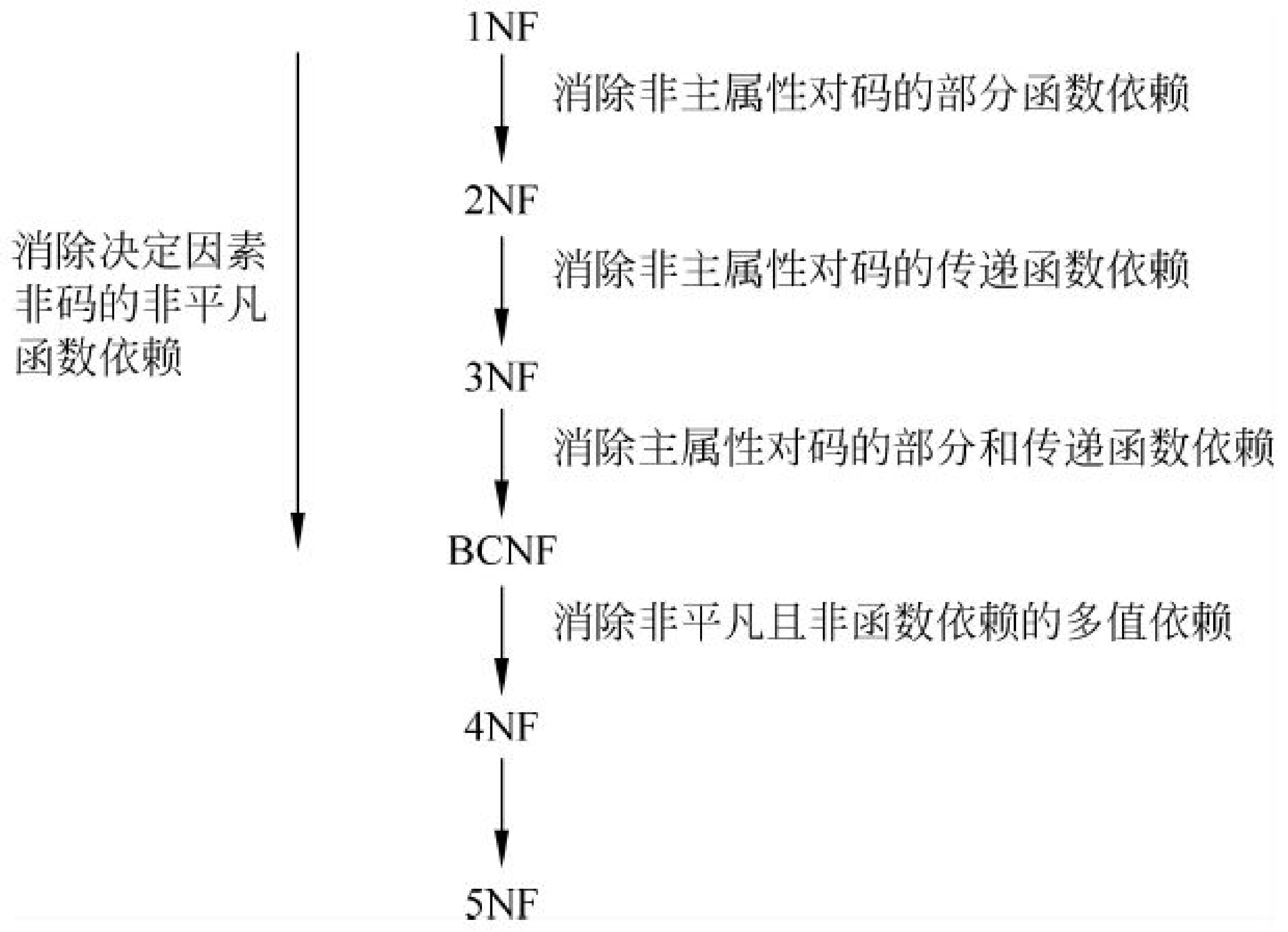

图 4-2　关系模式分解与规范化的关系

4.4　关系模式的分解

一个不合适的关系模式存在着数据存储和操作的问题，解决方法是将关系模式规范化，实际上，规范化过程是将一个关系模式分解成多个规范化的关系模式的过程。

4.4.1　关系模式分解的标准

一个关系模式 $R(U,F)$（其中 U 为 R 的属性的集合，F 为 R 的函数依赖集合）分解为若干个关系模式 $R_1(U_1,F_1)$，$R2(U_2,F_2)$，…，$R_K(U_K,F_K)$，其中 $U=U_1\cup U_2\cup\cdots\cup U_K$，且不存在 $U_i\subseteq U_j$，F_i 是 F 在 U_i 上的投影，这意味着相应地将存储在二维表 T 中的数据分散到若干个二维表 $T_1,T_2,\cdots,T_k$ 中去（其中 T_i 是 T 在属性组 U_i 上的投影）。我们希望这样的分解不丢失信息，也就是说，希望能通过对关系 $r_1,r_2,\cdots,r_k$ 的自然连接运算，重新得到关系 r 中的所有信息。

当然，在分解时关心的是分解后的关系模式是否能准确地反映原关系模式的所有信息，并且不会增加不存在的信息，即一方面要求分解后的关系模式经过某种连接操作后和原关系模式的记录相同，既没有增加也没有减少（无损连接），另一方面要求分解以后的关系模式保持了原关系模式中的函数依赖（保持依赖），这是关系模式分解的原则。

规范化过程中将一个关系模式分解为若干个关系模式，应该保证分解后产生的模式与原来的模式等价。通常关系模式分解等价标准是：①要求分解是具有无损连接性；②要求分解是保持函数依赖的。

4.4.2 无损连接性

设关系模式 $R(U,F)$（其中 U 为 R 的属性的集合，F 为 R 的函数依赖集合）分解为若干个关系模式 $R_1(U_1,F_1)$，$R_2(U_2,F_2)$，…，$R_K(U_K,F_K)$，其中 $U=U_1\cup U_2\cup\cdots\cup U_K$，且不存在 $U_i\subseteq U_j$，F_i 是 F 在 U_i 上的投影。若 R 与 R_1，R_2，R_K 自然连接的结果相等，则称关系 R 的这个分解具有无损连接性(lossless join)。

只有具有无损连接性的分解才能保证不丢失信息。

例如，关系模式 $R(A,B,C,D,E)$，其函数依赖关系为 $A\to B$，$A\to C$，$A\to D$，$C\to D$，$D\to E$。可以将 R 分解为 $R_1(A)$，$R_2(B)$，$R_3(C)$，$R_4(D)$ 和 $R_5(E)$，这一定是具有无损连接性的分解。但是，这不是一个好的分解，因为它没有保持函数依赖。

4.4.3 保持函数依赖

设关系模式 $R(U,F)$（其中 U 为 R 的属性的集合，F 为 R 的函数依赖集合）分解为若干个关系模式 $R_1(U_1,F_1)$，$R_2(U_2,F_2)$，…，$R_K(U_K,F_K)$，其中 $U=U_1\cup U_2\cup\cdots\cup U_K$，且不存在 $U_i\subseteq U_j$，F_i 是 F 在 U_i 上的投影。若 F 所逻辑蕴涵的函数依赖一定也由分解得到的某个关系模式中的函数依赖 F_i 所逻辑蕴涵，则称关系 R 的这个分解是保持函数依赖的(preserve dependency)。

例如，关系模式 $R(A,B,C,D,E)$，其函数依赖关系为 $A\to B$，$A\to C$，$A\to D$，$C\to D$，$D\to E$。将 R 分解，$R_1(A,B,C,D)$ 和 $R_2(D,E)$，这是保持函数依赖的分解，还可以进一步分解为 $R_1(A,B,C,D)$、$R_2(C,D)$ 和 $R_3(D,E)$，这也是保持函数依赖的分解，因为 R 的函数依赖集 $F=\{A\to B,A\to C,A\to D,C\to D,B\to E\}$，$R_1$ 的函数依赖集 $F_1=\{A\to B,A\to C\}$，R_2 的函数依赖集 $F_2=\{C\to D\}$，R_3 的函数依赖集 $F_3=\{D\to E\}$，F 所蕴涵的每个函数依赖关系均被 F_1、F_2、F_3 之一所蕴涵。

为了提高规范化程度，通常是通过将低一级的关系模式分解为若干个高一级的关系模式来实现，分解过程存在如下情况：

(1) 分解具有无损连接性和保持函数依赖性是两个互相独立的分解标准。具有无损连接性的分解不一定保持函数依赖；保持函数依赖的分解不一定具有无损连接性。

(2) 若要求分解具有无损连接性，那么模式分解一定可以达到 BCNF。

(3) 若要求分解保持函数依赖，那么模式分解可以达到 3NF，但不一定能达到 BCNF。

4.5 在实际数据库设计中关系规范化的应用

4.5.1 关系规范化的基本原则

一个低级范式的关系模式，通过关系模式的投影分解，可以转换为若干个高一级范式的关系模式的集合，这种过程就叫规范化。其基本思想是：逐步消除数据依赖中不合适的部分，使各关系模式达到一定程度的分离，即“一事一地”的模式设计原则，使概念单一化，即让一个关系描述一个概念、一个实体或者实体间的一种联系。

规范化的程度越高，数据的冗余和更新异常就越少，但由于连接运算费时，查询时所花的时间也就越多。因此，规范化宜根据具体情况权衡利弊，适可而止，对于数据变动不频繁的数据库，其规范化程度可以低一些。实际工作中，一般达到多数关系模式为 3NF 即可。

值得注意的是，规范化仅仅从一个侧面提供了改善关系模式的理论和方法。一个关系模式的好坏，规范化是衡量的标准之一，但不是唯一的标准。数据库设计者的任务是在一定的制约条件下，寻求能够较好地满足用户需求的关系模式。规范化的程度不是越高越好，这取决于应用。

根据关系数据库设计理论，优化关系数据库设计的过程，实际上就是对关系模式进行规范化的过程，即不断通过投影分解使非规范化的关系模式达到规范化的要求。前面已给出了许多分解关系模式的例子。一般地说，关系模式 $R(A_1, A_2, A_3, \cdots, A_n)$ 的分解就是用 R 的一组子集 $\{R_1, R_2, R_3, \cdots, R_k\}$ 来代替 R，且这组子集满足条件：

$$R = R_1 \cup R_2 \cup \cdots \cup R_k$$

其中，任意两个子集 R_j 和 R_i 相互的交不要求为空，即它们可以有共同的属性。通过分解，可消除数据冗余，从而消除插入、删除或更新的异常。对于关系分解，不仅要求消除数据冗余，还要求分解后的关系模式和分解前的关系模式能表示相同的信息。

在关系模式规范化时一般应遵循以下原则：

(1) 关系模式进行无损连接分解。关系模式分解过程中数据不能丢失或增加，必须把全局关系模式中的所有数据无损地分解到各个子关系模式中，以保证数据的完整性。

(2) 合理选择规范化程度。若考虑到存取效率，低级范式造成的冗余度很大，既浪费了存储空间，又影响了数据的一致性，因此希望一个子模式的属性越少越好，即取高级范式；若考虑到查询效率，低级范式又比高级范式好，此时连接运算的代价较小。这是一对矛盾，所以应根据实际情况，合理选择规范化程度。

(3) 正确性与可实现性原则。

4.5.2 关系规范化的实际应用

在上节提到的关系模式规范化的过程是：假定先用某种方法得到一个关系数据库的模式，然后分析和确定这个数据库模式上的所有存在的函数依赖，甚至包括多值依赖关系，再用相对机械的方法进行关系模式的分解，消除关系模式中的一些不适当的数据依赖关系，从而将关系数据库模式规范化。这是理论上的做法，而在实际应用中并不完全这样做。其主要原因在于：

第一，找出所有数据依赖关系不是一件简单的事，如果漏掉了或错误地确定了一些数据依赖关系，按上节的方法进行关系规范化，不能得到一个在理论上认为好的数据库设计。

第二，即使能正确地找出所有的数据依赖关系，采用机械地分解模式的方法，而完全不考虑关系的具体大小，以及数据的动态特征(比如数据是经常更新的，还是很少更新的)，全部规范化到同样的程度(比如 3NF)，这也是不适当的。

第三，数据库设计一般采用先得到现实环境的 E-R 模型，再由 E-R 模型转换得到关系数据库模式的方法。从 E-R 模型转换生成的关系模式，很少有较大关系模式(包含属性较多)，由于 E-R 模型中往往实体集合分得较细，转换得到的关系模式较小(包含属性较少)，

为了以后数据库查询的方便，更多的情况是需要合并关系模式，而不是分解关系模式。因此，在实际应用中，要得到一个好的数据库设计，需要根据具体情况对关系模式进行处理，既可能要合并关系模式，也可能要分解关系模式。

当然，这并不意味着关系规范化的理论在实际的数据库设计中就没有意义，它对关系数据库模式的设计仍然起着指导作用。当设计 E-R 模型时，以及由 E-R 模型转换生成关系模式后，在进行关系模式优化设计的过程中，关系的规范化理论能够帮助用户得到较好的数据库设计。在设计 E-R 模型时，应仔细分析数据间存在的函数依赖和多值依赖，这样能使用户最后从 E-R 模型得到的关系数据库模式基本达到 3NF 的规范程度。

数据之间的函数依赖是现实世界中客观存在的，数据依赖关系的确定最好在系统分析期间，在生成 E-R 模型的过程中完成，而不必在已经得到关系数据库模式后，再去寻找有哪些数据依赖关系存在。实际上，在生成 E-R 模型时确定数据之间的依赖关系更为容易，因为尽管 E-R 模型本身并不包括函数依赖和多值依赖等概念，但 E-R 模型更接近于现实世界。例如，当确定了一个实体集合和它的属性后，自然就确定了属性对实体集合的依赖关系，以及实体集合非关键字属性对实体集合关键字属性的依赖关系；类似地，如果联系 R 表示从实体集合 $E1$ 到实体集合 $E2$ 的多对一联系，广义地，可以说实体集合 $E1$ 决定了实体集合 $E2$，在转化为关系模式后，联系 R 形成的关系模式的属性集合中有 $E1$ 的关键字 X 和 $E2$ 的关键字 Y，则有 $X \rightarrow Y$ 成立，而且 X 决定了 R 的任何一个属性集，若 $E1$ 和 $E2$ 之间的联系是一对一的，则 $Y \rightarrow X$ 也成立。所以，在 E-R 模型中是隐含着许多数据依赖关系的。

在 E-R 模型中，任何事物、数据或知识等都可以是实体，实体的属性是实体某一方面的特征，它完全可能有非常复杂的结构，而且属性之间可能存在各种各样的数据依赖关系。由于建立 E-R 模型的目的是最终生成关系数据库模式，而在关系数据库模式中不能描述复杂结构的属性，因此，当发现实体有结构复杂的属性时，通常要在 E-R 模型中加入新的实体来解决这个问题。这就相当于将非 NF 的关系模式规范为 1NF 的关系模式的工作，但这里通常不是把嵌套的属性(或称为子表)“展开”，使嵌套属性成为原子属性，而是将一个实体“分解”成多个新实体，将具有复杂结构的属性处理成实体。类似地，当发现一实体内的属性之间存在除对关键字属性的完全函数依赖以外的函数依赖，或者分析插入、删除等动态特性发现有可能发生异常时，也试图发现新实体，并将其增加到 E-R 模型中，以消除这些函数依赖。当然，是否一定需要消除这些函数依赖，还要综合考虑数据冗余和数据的动态特性(做插入、删除、修改的频率)，因为函数依赖多，数据冗余就多，但数据的查询代价小。一般，凡是因为保留数据依赖而造成数据冗余的，都应设计数据库触发器或其他设施，来保证在插入、删除、修改时数据的一致性，以避免任何异常的出现。事实上，只要在进行系统需求分析时，对企业或组织的分析很全面、细致和正确，不遗漏任何实体，就有可能消除“多余”的函数依赖，得到 E-R 模型转化成的关系模式基本上是 3NF。

从 E-R 模型转化为关系模型后，常合并一些关系以求模型简单、易用。合并关系时应注意不要产生“多余”的函数依赖，以免造成数据冗余。合并关键字相同的关系模式不会产生数据冗余；合并存在外部关键字约束的两个关系模式，则会产生数据冗余，因此是否合并就应做综合权衡。

4.6 本章小结

本章讨论关系数据理论。要设计好的数据库模式,必须有模式设计理论为基础。规范化设计理论对关系数据库结构的设计起着重要的作用。

函数依赖是对关系中属性值之间多对一的描述,也是对关系中值的一种约束。它是对关键码概念的扩充。

范式是衡量模式优劣的标准。范式的级别越高,其数据冗余和操作异常现象就越少。

关系模式的规范化过程就是把低一级的关系模式通过模式分解,转化为若干个高一级范式的关系模式的集合的过程。其目的是解决关系模式中存在的插入、删除异常、修改复杂,数据冗余等问题。

关系模式在分解时,应该保证分解后产生的模式与原来的模式等价。通常关系模式分解等价分别用无损连接性和保持函数依赖两个特征来衡量。

4.7 习题

4.7.1 名词解释

关系模式的规范化、函数依赖、部分函数依赖、完全函数依赖、传递依赖、候选码、主码、外码、全码、1NF、2NF、3NF、BCNF

4.7.2 简答题

1. 数据存储异常包括哪几类?
2. 简述关系无损连接性和保持函数依赖之间的联系。

4.7.3 综合题

1. 设关系模式 $R(A,B,C,D)$,其函数依赖关系为$\{A\to C, C\to A, B\to AC, D\to AC, BD\to A\}$,求出关系模式 R 的候选码;将 R 分解为 3NF,使其既具有无损连接性又具有函数依赖保持性。

2. 设关系模式 $R(A,B,C,D,E,F)$,其函数依赖关系 $F=\{AB\to E, BC\to D, BE\to C, CD\to B, CE\to AF, CF\to BD, C\to A, D\to EF\}$,求 F 的最小函数依赖集。

3. 设关系模式 $R(A,B,C,D)$,其函数依赖关系 $F=\{AB\to CD, A\to D\}$,试说明 R 不是 2NF 模式的理由,并把 R 分解成 2NF 模式集。

4. 设有关系模式 R(员工名,项目名,工资,部门名,部门经理),规定每个员工可参加多个项目建设,各领一份工资;每个项目只属于一个部门管理;每个部门只有一个经理。

(1) 写出关系模式 R 的基本函数依赖和关键码。

(2) 说明 R 不是 2NF 模式的理由,并把 R 分解成 2NF 模式集。

(3) 进而把 R 分解成 3NF 模式集,并说明理由。

5. 设有关系模式 R(员工名,项目名,工资,部门名,部门经理),规定每个员工可参加多个项目建设,各领一份工资;每个项目只属于一个部门管理;每个部门只有一个经理。

(1) 试写出关系模式 R 的基本函数依赖和关键码。

(2) 说明 R 不是 2NF 模式的理由,并把 R 分解成 2NF 模式集。

(3) 进而把 R 分解成 3NF 模式集,并说明理由。

6. 判断下面的关系模式是不是 BCNF,并说明理由。

(1) 任何一个二元关系;

(2) 关系模式 $R(A,B,C,D,E,F)$,函数依赖集$\{A\rightarrow BC, BC\rightarrow A, BCD\rightarrow EF, E\rightarrow C\}$。

7. 设有关系模式 $R(A,B,C)$,其函数依赖关系 $F=\{AB\rightarrow C, C\rightarrow\rightarrow A\}$,$R$ 属于第几范式,为什么?

8. 下列结论哪些是正确的?哪些是错误的?对于错误的请给出一个反例。

(1) 任何一个二元关系都是属于 3NF 的。

(2) 任何一个二元关系都是属于 BCNF 的。

(3) 任何一个二元关系都是属于 4NF 的。

第5章 关系查询处理与优化

本章主要介绍关系型数据库系统的查询优化技术。查询优化技术在关系型数据库中有着非常重要的作用，关系型数据库系统和非过程化 SQL 语言的巨大成功，得益于查询优化技术的发展，查询优化是影响 RDBMS 性能的重要因素，也是其优势所在。本章将介绍关系型数据库的查询优化方法及相关问题，将使读者初步了解 RDBMS 查询的基本处理过程，以及查询优化的基本概念和基本方法，并给出了一些在实际应用中的查询优化方法。

5.1 查询优化概述

查询优化对关系型系统来说既是挑战又是机遇。所谓挑战是指关系系统为了达到用户可接受的性能必须进行查询优化。由于关系表达式的语义级别很高，使得关系系统能够从关系表达式中分析查询语义，提供了查询优化的可行性。这为关系系统在性能上接近甚至超过非关系系统提供了机遇。

本节主要讨论查询优化问题的背景、查询优化的必要性和在关系数据库中进行查询的可行性。

5.1.1 查询中遇到的问题

数据查询是数据库系统中的最基本、最常用和最复杂的数据操作，从实际应用的角度看，必须考察系统用于查询处理的开销代价。查询处理的代价通常取决于查询过程对磁盘的访问，磁盘访问速度相对于内存速度要慢很多。在数据库系统中，用户的查询通过相应的查询语句提交给 DBMS 执行。一般而言，相同的查询要求和结果存在着不同的实现策略，在执行这些查询策略时系统所付出的开销代价通常有很大差别，甚至可能会相差几个数量级。实际上，对于任何一个数据库系统来说，查询处理都是必须要面对的，如何从查询的多个实现策略中进行合理的选择，这种选择过程就是查询处理过程的优化，简称查询优化。

查询优化作为数据库中的关键技术，对数据库的性能需求和实际应用有着重要的意义。

查询是数据库的最主要的功能；数据查询必然会有查询优化的问题；从对数据库的性能要求和使用技术的角度来看，在任何一种数据库中都要有相应的处理方法和途径。查询优化的基本途径可以分为用户手动处理和机器自动处理两种。

关系型数据库系统中，查询优化也是必须面对的挑战。关系数据理论基于集合论，集合及其相关理论构成了整个关系数据库领域中最重要的理论基础，这给关系数据查询优化处

理在理论上提供了讨论的可行性；关系查询语言作为高级语言，具有较高层次的语义特性，为机器处理查询优化问题在实践上提供了可能性。

5.1.2 查询优化的必要性

一个好的查询计划往往可以使程序性能提高数十倍，查询计划是用户所提交的 SQL 语句的集合。DBMS 处理查询计划的过程是这样的：在做完查询语句的词法、语法、语义检查之后，将语句提交给 DBMS 的查询优化器，优化器做完代数优化和存取路径优化之后，由预编译模块对语句进行处理并生成查询规划，然后在合适的时间提交给系统处理执行，最后将执行结果返回给用户。

下面通过一个例子来说明查询优化的必要性。

【例 5.1】 在关系模式 S(学生)，C(课程)，SC(选课)中，查询修读课程号为 $C5$ 的所有学生姓名。

此查询的 SQL 查询语言的语句形式为：

```
SELECT S.Sname FROM S,SC WHERE S.Sno = SC.Sno AND SC.Cno = 'C5';
```

系统可以有多种等价的关系代数表达式来完成这一查询。一般而言，在 SQL 语句转换为关系代数表达式的过程中，SELECT 语言对应投影运算，FROM 语句对应笛卡儿乘积运算，WHERE 子句对应选择运算。例如可以写出下面 3 种表达式：

$$Q_1 = \pi_{\text{Sname}}(\sigma_{S.\text{Sno}=SC.\text{Sno} \wedge SC.\text{Cno}=\text{C5}}(S \times SC));$$

$$Q_2 = \pi_{\text{Sname}}(\sigma_{SC.\text{Cno}=\text{C5}}(S \bowtie SC));$$

$$Q_3 = \pi_{\text{Sname}}(S \bowtie \sigma_{SC.\text{Cno}=\text{C5}}(SC))。$$

当然，还可以写出其他等价的关系代数表达式，但只要分析以上式子就可以说明问题。

下面用简单的方法来计算这 3 种表达式查询所需的时间。在计算前先做如下的统一约定：

- 设 S 有 1000 个元组，SC 有 10000 个元组，其中修读 $C5$ 的元组数为 50。
- 磁盘中每个物理块能存放 10 个 S 元组，或 100 个 SC 元组。
- 内存有 6 个块的缓冲区，其中 5 块可以存放 S 元组，1 块存放 SC 元组。
- 读/写一块磁盘的时间为 1/20s，即 1s 读写 20 个磁盘块。
- 为了简化起见，所有内存操作所花的时间忽略不计。

1. 计算 Q_1 的查询时间

(1) 首先做笛卡儿乘积

将 S 与 SC 的每个元组相连接，其方法为先读入 S 中的 50 个元组(5×10)至 S 表中的内存缓冲区，然后不断地将 SC 的元组按 100 位一块读入后与 S 的元组相连接，直至读完所有 SC 元组(共计 100 次)。这种操作内连接满 100 位后就写中间文件一次。反复进行这样的操作，直至做完笛卡儿乘积，此时共读取总块数为：

$$\frac{1000}{10} + \frac{1000}{10 \times 5} \times \frac{10\,000}{100} = 100 + 20 \times 100 = 2100(\text{块})$$

其中读 S 表 100 块，读 SC 表 20 次，每次 100 块。由于每块花费时间 1/20s，此时总共

花费时间 105s。连接后的元组数为 $10^3\times10^4=10^7$，设每块(约)能装 10 个元组，则写入中间文件要花 $10^6/20=5\times10^4$ s。

(2) 其次做选择操作

从中间文件中读出连接后的元组，按选择要求选取记录(此项为内存操作，时间可忽略不计)，此项操作所需时间与写入中间文件时间一样，即 5×10^4 s。满足条件的元组假设为 50 个，均放在内存。

(3) 最后做投影操作

第二项操作的结果满足条件的元组数为 50 个，它们可全部存放在内存。对它们在 S 上做投影操作，由于是在内存中进行，其时间可忽略不计。这样 Q_1 的全部查询时间为 $10+2\times5\times10^4\approx10^5$ s。注意到一天为 86 400s，所以这个运算需要超过一天的时间来完成。

2. 计算 Q_2 的查询时间

- 计算自然连接：计算自然连接时读取 S 与 SC 表的方式与 Q_1 一致，总读取块数为 2100 块，花费时间为 105s，但其连接结果块数大为减少，总计 10^4 个，所花时间为 $10^4/10/20$s＝50s，仅为 Q_1 的千分之一。
- 做选择操作：做选择操作的时间为 50s。
- 做投影操作：与 Q_1 类似，其时间可忽略不计。

这样，Q_2 的全部查询时间为：

$$105+50+50=205\text{s}$$

3. 计算 Q_3 的查询时间

- 对 SC 做选择操作：对 SC 表做选择操作需读 SC 表一遍共计读 100 块，花费 5s，因为满足条件的元件只有 50 个，不必使用中间文件。
- 做连接运算：对 S 选择后的 SC 左联接运算，由于选择后的 SC 已全部在内存，因此全部操作时间为 S 读入内存的时间共 100 块，花费时间为 5s。
- 做投影运算：其时间忽略不计。

这样，Q_3 的全部查询时间为：

$$5+5=10\text{s}$$

从这 3 个计算时间可以看出，3 种等价的查询表达式具有完全不同的处理时间，它们分别是 10^5 s、205s 和 10s，其差距之大令人瞠目。

由以上示例可知，对于关系代数等价的不同表达形式而言，相应的查询效率有着“数量级”上的重大差异。这是一个十分重要的事实，它说明了查询优化的必要性，即合理选取查询表达式可以获取较高的查询效率，这也是查询优化的意义所在。

5.1.3　查询优化的可行性

我们知道，关系数据系统查询语句表示查询操作基于集合运算，一般称之为关系代数。关系代数具有 5 种基本运算，这些运算间满足一定的运算定律，如结合律、交换律、分配率和串接率等，这就意味着不同的关系代数表达式可以得到同一结果，因此用关系代数语言进行查询需要进行必要的优化。

关系查询语句与普通语言相比有坚实的理论支撑，人们能够找到有效的算法，使查询优化的过程内含于 DBMS，由 DBMS 自动完成，从而将实际上的“过程性”向用户“屏蔽”，用户只需提出“干什么”，而不必指出“怎么干”，这样用户在编程时只需表示出所需要的结果，不必给出获得结果的操作步骤。从这种意义上讲，关系查询语言是一种高级语言，这给查询优化提供了可能性。

5.2 查询处理过程

在关系型数据库中，查询处理过程通常可以分为四个阶段：查询分析、查询处理、查询优化和查询执行，如图 5-1 所示。

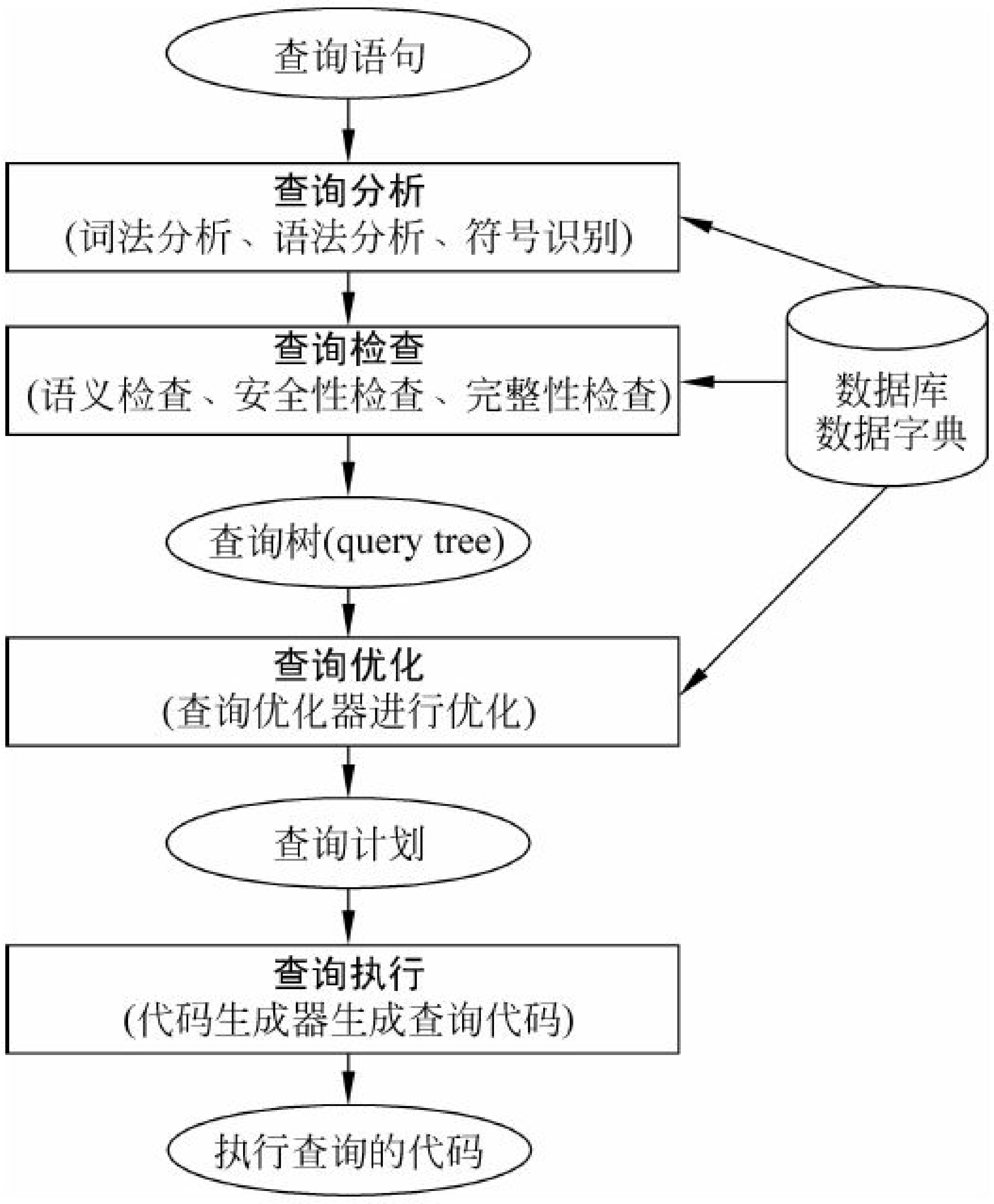

图 5-1 关系数据库查询处理步骤

5.2.1 查询分析

查询分析要对查询语句进行扫描、词法分析和语法分析。从查询语句中识别出语言符号，如 SQL 关键字、属性和关系名称等，并进行语法检查和分析，检查语句是否符合 SQL 语法规则。

5.2.2 查询检查

查询检查首先根据数据字典对合法的查询语句进行语义检查，检查语句中识别出的语言符号在数据库中是否存在，是否有效。还要根据数据字典中的用户权限和完整性约束定义对用户进行检查，如果用户不具备相应的访问权限或者违反了完整性约束原则，就拒绝执

行该查询。检查通过后，把查询语句转化成为等价的关系代数表达式。在 RDBMS 中一般都用查询树(query tree)(也称为语法分析树(syntax tree))，来作为查询的内部表示形式。

5.2.3 查询优化

每个查询都会有多种可供选择的执行策略和操作算法，查询优化就是选择一个高效的查询处理策略。DBMS 会调用系统的优化处理器制定一个执行策略，由此产生一个查询计划，其中包括了如何防伪数据库文件和如何存储中间结果等。

对于关系型数据库来说，查询优化过程是由 RDBMS 自动完成的，它与用户书写的查询语句无关。系统自动生成的若干候选查询计划并从中选取"好的"查询计划的程序称为查询优化器。查询优化器是关系数据库的巨大优势所在，它使用户不必考虑如何较好地表达查询以获得较高的效率，而且系统自动优化存取路径可以比用户的程序做得更好。

查询优化有多种方法。①代数优化：指关系代数表达式的优化，即根据某些启发式规则，改变代数表达式中的次序和组合，使查询执行得更高效，例如"先选择、投影和后连接"等就可完成优化，所以也成为规则优化；②物理优化：存取路径和底层操作算法的选择，可以是基于规则的、基于代价的，也可以是基于语义的。

实际优化过程中为了达到更好的优化效果往往都综合使用这些优化技术。

5.2.4 查询执行

针对查询优化器得到的查询计划，系统的代码生成器产生出这个计划的执行代码。

5.3 查询优化方法

5.3.1 代数优化

前面已经介绍过，查询语句经过查询分析、查询检查后变为查询树，它是代数表达式的内部表示形式。代数优化主要是依据关系代数表达式的等价变换规则所做的优化。

代数优化策略是通过对代数关系表达式的等价变换来提高效率的。关系代数表达式的等价是指两个关系代数表达式用同一个关系代入之后得到的结果相同，即两个相应的关系具有相同的属性结合和相同的元组集合。

1. 关系代数表达式变换规则

常用的代数表达式的等价变换规则主要有以下几类，证明从略。

(1) 连接、笛卡儿积的交换律

设 E_1 和 E_2 是关系代数表达式，F 是连接运算条件，则下列等价公式成立：

- 笛卡儿积的交换律

$$E_1 \times E_2 \equiv E_2 \times E_1$$

- 自然连接的交换律

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

• 条件连接的交换律

$$E_1 \underset{F}{\bowtie} E_2 \equiv E_2 \underset{F}{\bowtie} E_1$$

(2) 连接、笛卡儿积的结合律

设 E_1、E_2 和 E_3 是关系代数表达式，F_1 和 F_2 是连接运算条件，则下列等价公式成立：

• 笛卡儿积的结合律

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

• 自然连接的结合律

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

• 条件连接的结合律

$$(E_1 \underset{F_1}{\bowtie} E_2) \underset{F_2}{\bowtie} E_3 \equiv E_1 \underset{F_1}{\bowtie} (E_2 \underset{F_2}{\bowtie} E_3)$$

(3) 选择、投影的串接定律

① 选择运算串接定律

设 E 是一个关系代数表达式，F_1 和 F_2 是选择运算条件，则下列等价公式成立：

• 选择运算顺序可交换定律

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_2}(\sigma_{F_1}(E))$$

• 合取条件分解定律

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_2 \wedge F_1}(E)$$

② 投影运算串接定律

设 E 是关系代数表达式，$A_i(i=1,2,\cdots,n)$，$B_j(j=1,2,\cdots,m)$ 是 E 中的某些属性名，且 $\{A_1,A_2,\cdots,A_n\}$ 是 $\{B_1,B_2,\cdots,B_m\}$ 的子集，则下列等价公式成立：

$$\pi_{A_1,A_2,\cdots,A_n}(\pi_{B_1,B_2,\cdots,B_m}(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(E)$$

(4) 选择与笛卡儿积、选择与投影的交换律

• 选择与笛卡儿积的交换律

设 E_1、E_2 是关系代数表达式，如果 F 中涉及的属性都是 E_1 中的属性，则下列等价公式成立：

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$$

如果 $F = F_1 \wedge F_2$，并且 F_1 只涉及 E_1 中的属性，F_2 只涉及 E_2 中的属性，则有上述规则可推出：

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times \sigma_{F2}(E_2)$$

若 F_1 只涉及 E_1 中的属性，F_2 涉及 E_1 和 E_2 两者的属性，则有：

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F2}(\sigma_{F1}(E_1) \times E_2)$$

这样可以使部分选择运算在笛卡儿积前先做。

• 选择与投影的交换律

设 E 是关系代数表达式，$A_i(i=1,2,\cdots,n)$、$B_j(j=1,2,\cdots,m)$ 是 E 的属性，A_i 与 B_j 不相交，F 是选择条件。如果 F 只涉及 $\{A_1, A_2,\cdots, A_n\}$，则下列等价公式成立：

$$\sigma_F(\pi_{A_1,A_2,\cdots,A_n}(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E))$$

如果 F 中有不属于 $\{A_1, A_2,\cdots, A_n\}$ 的属性 $\{B_1, B_2,\cdots, B_m\}$，则有更一般的规则：

$$\pi_{A_1,A_2,\cdots,A_n}(\sigma_F(E)) \equiv \pi_{A_1,A_2,\cdots,A_n}(\sigma_F(\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E)))$$

(5) 选择与并运算、选择与差运算、选择与自然连接的分配率

• 选择与并运算的分配率

设 E_1 和 E_2 是两个关系代数表达式，并且 E_1 和 E_2 具有相同的属性名，F 是选择条件，则下列等式成立：

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

• 选择与差运算的分配率

设 E_1 和 E_2 是两个关系代数表达式，并且 E_1 和 E_2 具有相同的属性名，F 是选择条件，则下列等式成立：

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

• 选择与自然连接的分配率

设 E_1 和 E_2 是两个关系代数表达式，并且 E_1 和 E_2 具有相同的属性名，F 是选择条件，F 只涉及 E_1 与 E_2 的公共属性，则下列等式成立：

$$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$$

(6) 投影与笛卡儿积、投影与并的分配率

• 投影与笛卡儿积的分配率

设 E_1 和 E_2 是两个关系代数表达式，$A_i(i=1,2,\cdots,n)$是 E_1 的属性，$B_j(j=1,2,\cdots,m)$是 E_2 的属性，则下列等价公式成立：

$$\pi_{A_1,A_2,\cdots,A_n,B_1,B_2,\cdots,B_m}(E_1 \times E_2) \equiv \pi_{A_1,A_2,\cdots,A_n}(E_1) \times \pi_{B_1,B_2,\cdots,B_m}(E_2)$$

• 投影与并的分配率

设 E_1 和 E_2 是两个关系代数表达式，$A_i(i=1,2,\cdots,n)$是 E_1、E_2 的公共属性，则下列等价公式成立：

$$\pi_{A_1,A_2,\cdots,A_n}(E_1 \cup E_2) \equiv \pi_{A_1,A_2,\cdots,A_n}(E_1) \cup \pi_{A_1,A_2,\cdots,A_n}(E_2)$$

2. 查询树的启发式规则

本节讨论的是应用启发式规则的代数优化。通过启发式规则对关系表达式的查询树表达形式进行优化。典型的启发式规则如下。

(1) 选择操作优先原则

及早进行选择操作，这是查询优化中的最重要最基本的一条。尽可能地先做选择操作，常常会将查询执行时间降低好几个数量级，因为选择操作会使计算的中间结果大大变小。

(2) 投影操作优先原则

及早进行扫描操作，可以与相关的元素按合并进行。例如可以把若干针对同一个关系操作的投影与选择运算合并进行，这样可以避免重复扫描关系，在笛卡儿积或连接操作之前尽量排除无关数据。

(3) 笛卡儿积合并规则

尽量避免单独进行笛卡儿积操作，可以把笛卡儿积与之前和之后的一系列选择和投影运算合并起来一起操作，这样可以避免两次操作之间的磁盘存取，特别是笛卡儿乘积大量的中间数据。

(4) 提取公共表达式规则

如果重复出现的子表达式不是很大的关系，并且存磁盘中读取这个关系比计算这个表达式的时间要少得多，则应先计算一次此表达式，把中间结果存储起来，下次使用时直接读

取这个结果，这样可以减少计算，提高效率。

下面给出遵循这些启发式规则，应用等价变换公式来优化关系表达式的算法。

算法：关系表达式的优化。

输入：一个关系表达式的查询树。

输出：优化的查询树。

方法：

① 利用等价变换规则把形如 $\sigma_{F_1 \wedge F_2 \wedge \cdots \wedge F_n}(E)$ 变换为 $\sigma_{F_1}(\sigma_{F_2}(\cdots(\sigma_{F_n}(E))\cdots))$。

② 对每一个选择，利用等价变化规则，尽量把它移到树的叶端。

③ 对每一个投影利用等价变换规则，尽可能把它移向树的叶端。

④ 利用等价变换规则，把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影。使多个选择或投影能同时执行，或在一次扫描中全部完成，尽管这种变换似乎违背了“投影尽可能早做”的原则，但这样做效率更高。

⑤ 把上述得到的语法树的内结点分组。每一双目运算（×，⋈，∪，－）和它所有的直接祖先为一组，这些直接祖先是（σ，π 运算）。如果其后代直到叶子全是单目运算，则也将它们并入该组，但当双目运算是笛卡儿积（×），而且后面不是与它组成等值连接的选择时，则不能把选择与这个双目运算组成同一组，应当把这些单目运算单独分为一组。

【例 5.2】 下面给出了一个 SQL 语句的代数优化示例。

求选修了 02 号课程的学生姓名。用 SQL 表达：

```
SELECT S.Sname FROM S,SC WHERE S.Sno = SC.Sno AND SC.Cno = '02';
```

(1) 把 SQL 语句转换成查询树，如图 5-2 所示。

为了使用关系代数表达式的优化法，不妨假设内部表示是关系代数语法树，则上面的查询树如图 5-3 所示。

(2) 对查询树进行优化。

利用转换规则把选择 $\sigma_{SC.Cno=2}$ 移到叶端，图 5-3 所示的查询树便转换成图 5-4 所示的优化的查询树。

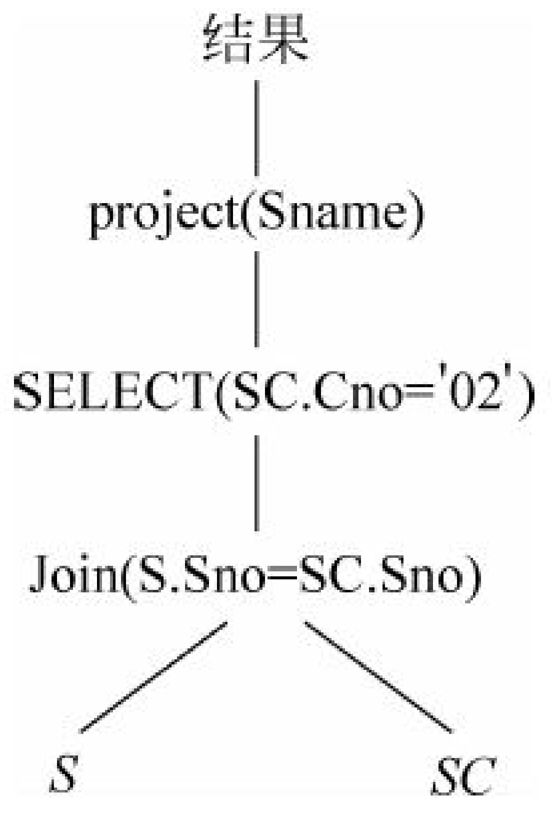

图 5-2 查询树

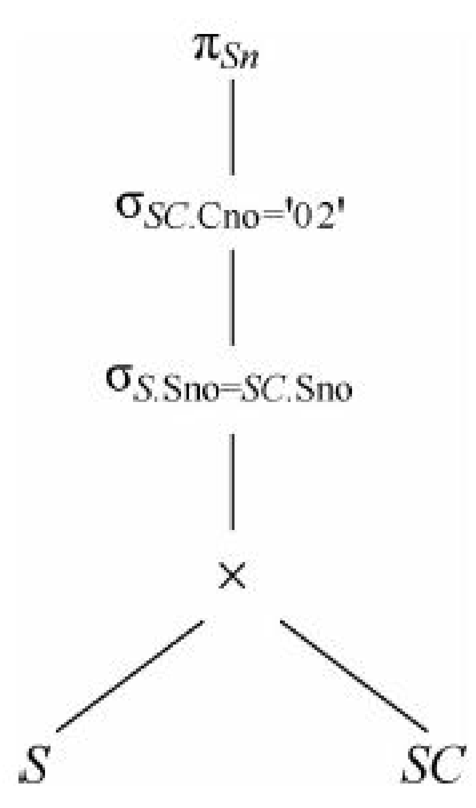

图 5-3 关系代数语法树

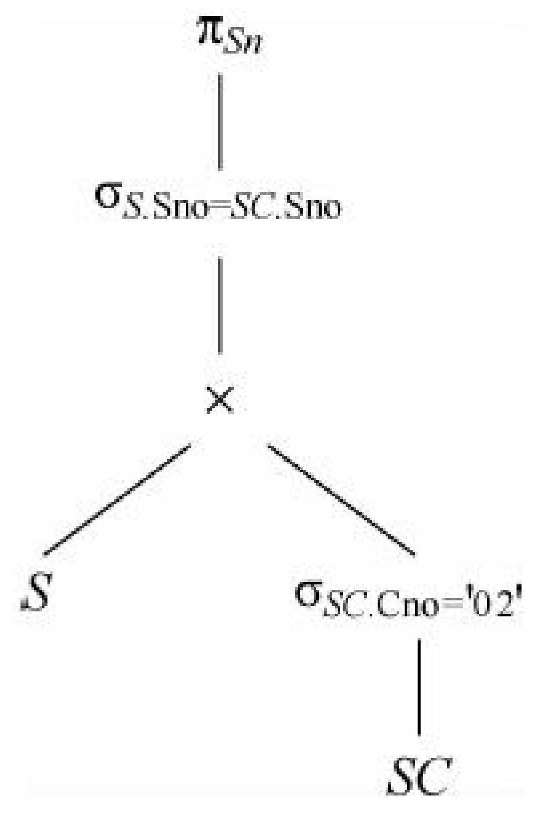

图 5-4 优化后的查询树

5.3.2 物理优化

代数优化改变了查询语句中操作的次序和组合方式，但不涉及底层存取路径。而物理优化主要是选择合理高效的操作算法或存取路径，得到优化的查询计划，来达到查询优化的目的。

选择的方式可以是:

- 基于启发式规则的优化
  启发式规则是指那些在大多数情况下都适用,但并不是在每种情况下都适用的规则。
- 基于代价估计的优化
  由优化器估算每种不同执行策略的代价,并选出具有最小代价的执行计划。
- 两者相结合的优化方法
  查询优化器常会把这两种技术结合在一起使用。因为能执行的策略有很多,要穷尽所有的查询策略进行代价估算往往是不可行的,会使查询优化本身付出的代价大于获得的益处。为此常常先使用启发式规则,选取若干较优的候选方案以减少代价评估的工作量;然后分别计算各方案的执行代价,从中较快地选择出最终的优化方案。

1. 基于启发式规则的存取路径选择优化

(1) 选择操作的启发式规则

① 对于小关系,使用全表扫描,即使列上有索引。

② 对于大关系:

- 对于选择条件是"主码=值"的查询,查询结果最多是一个元组,可以选择主码索引。一般的 RDBMS 会自动创建主码索引。
- 对于选择条件是"非主属性=值"的查询,并且选择列上有索引,则要估算查询结果元组的数目,如果比例较小(<10%)可以使用索引扫描方法,否则应该使用全表扫描。
- 对于选择条件是属性上的非等值查询或者范围查询时,并且选择列上有索引,同样也需要估算查询结果的元组数目,如果比例较小(<10%)则可以使用索引扫描方法,否则还是应该使用全表扫描。
- 对于用 AND 连接的合取选择条件,如果涉及这些属性的组合索引,则优先采用组合索引扫描方法。
- 对于用 OR 连接的析取选择条件,一般使用全表顺序扫描。

(2) 连接操作的启发式规则

① 如果两个表都已经按照连接属性排序,则选用排序-合并方法。

② 如果一个表在连接属性上有索引,则可以选用索引连接方法。

③ 如果上面两条规则都不适用,其中一个表较小,则可选用 Hash join 方法。

④ 最后可以选用嵌套循环方法,并选用其中较小的表(确切地说是占用的块数较少的表)作为外层循环的表。

上面仅仅列出了一些主要的启发式规则,在适应的 RDBMS 中启发式规则要多得多。

2. 基于代价估计的优化

启发式优化规则是定性选择,比较粗糙,但是实现简单而且优化本身代价较小,适合于解释执行系统。因为解释执行的系统,优化开销包含在查询总开销之中。而在编译执行的系统中,一次编译优化多次执行,查询优化和查询执行是分开的,因此,可以采用更精细更复

杂的机遇代价的优化方法。

(1) 信息统计

基于代价的优化方法就是要计算各种操作算法的执行代价，而代价的计算算法与数据库的状态密切相关。为此在数据字典中存储了优化器需要的统计信息(database statistics)，这些信息主要有：

① 基本表

对每个基本表，包含该表的元组总数(N)、元组长度(l)、占用的块数(B)、占用的溢出块数(BO)等信息。

② 基本表的列

对基表的每个列，包含该列不同值的个数(m)、选择率(f)、该列最大值、最小值，该列上是否已经建立了索引，及索引的种类。

③ 索引

对索引，例如B+树索引，该索引的层数(L)、不同索引值的个数、索引的选择基数S(有S个元组具有某个索引值)、索引的叶结点数(Y)。

除了上面的这些信息，还有一些必要的其他信息。

(2) 估算代价

在上面这些信息的基础上查询优化器可以采取一些算法来估计执行查询的代价。下面给出一些常用的操作算法的执行代价估算方法。

① 全表扫描算法代价的估算

如果基本表大小为B块，则全表扫描算法的代价 cost$=B$；

如果选择条件是“码＝值”，那么平均搜索代价 cost$=B/2$。

② 索引扫描算法代价的估算

如果选择条件是“码＝值”，则采用该表的主索引，若为B+树，层数为L，需要存取B+树中从根结点到叶结点L块，再加上基本表中该元组所在的那一块，所以 cost$=L+1$。

如果选择条件涉及非码属性，若为B+树索引，选择条件是相等比较，S是索引的选择基数(有S个元组满足条件)。因为满足条件的元组可能会保存在不同的块上，所以(最坏的情况)cost$=L+S$。

如果比较条件是非等值操作，假设有一半的元组满足条件，那么就要存取一半的叶结点，并通过索引访问一半的表存储块。所以 cost$=L+Y/2+B/2$。如果可以获得更准确的选择基数，可以进一步修正$Y/2$与$B/2$。

③ 嵌套循环连接算法代价的估算

嵌套循环连接算法的代价为 cost＝Br＋Br Bs/($K-1$)。如果需要把连接结果写回磁盘，则 cost＝ Br＋ Br Bs/($K-1$)＋(Frs ∗ Nr ∗ Ns)/Mrs。其中 Frs 为连接选择性(join selectivity)，表示连接结果元组数的比例，Mrs 是存放连接结果的块因子，表示每块中可以存放的结果元组数目。

④ 排序-合并连接算法代价的估算

如果连接表已经按照连接属性排好序，则 cost＝ Br＋Bs＋(Frs ∗ Nr ∗ Ns)/Mrs。

如果必须对文件排序，那么还需要在代价函数中加上排序的代价。对于包含B个块的文件排序的代价大约是$(2\times B)+(2\times B\times \log_2 B)$。

上面仅仅列出了少数操作算法的代价估算示例。在实际的RDBMS中代价估算公式要多得多，也复杂得多。

5.4 实际应用中的查询优化

本小节将从实际应用层面讨论在使用数据库时可以采取哪些策略来提高查询优化的效率。

5.4.1 基于索引的优化

一般来说，建立和删除索引的工作都是由数据库管理员DBA或表的主人（owner）即建立表的主人负责完成的。系统在存取数据时会自动选择合适的索引作为存取路径，用户不必显式地选择索引。

使用索引的一个主要目的是避免全表扫描，减少磁盘I/O，加快数据库查询的速度。但是，索引的建立降低了数据更新的速度，因为数据不仅要增加到表中，而且还要增加到索引中。另外，索引还需要额外的磁盘空间和维护开销。所以在设计和使用索引时应仔细考虑实际应用中修改和查询的频率，权衡建立索引的利弊，并应遵循以下原则：

（1）值得建索引并且用得上。记录有一定规模，而且查询只限于少数记录，规模小的表不宜建立索引。索引在WHERE子句中应频繁使用，索引并不是越多越好，当对表执行更新操作时系统会自动更新该表的所有索引文件，更新索引文件是要耗费时间的，因此也就降低了系统的效率。

（2）先装数据，后建索引。对于大多数基本表，总是有一批初始数据需要装入。建立关系后，先将这些初始数据装入基本表，然后再建立索引，这样可以加快初始数据的录入速度。

（3）查询语句中经常进行排序或在分组（用GROUP BY或ORDER BY操作）的列上建立索引。如果待排序的列有多个，可以在这些列上建立复合索引。但应注意在建立复合索引时涉及的属性列不要太多，否则会增加索引的开销。当执行更新操作时会降低数据库的更新速度。组合索引前导列在查询条件中必须使用，否则该组合索引失效。如果SQL中能形成索引覆盖，性能将达到最优。

（4）需要返回某字段局部范围的大量数据，应在该字段建立聚簇索引。经常修改的列不应该建立聚簇索引，否则会降低系统的运行效率。在经常进行连接，但没有指定为外键的列上建立索引，而不经常连接的字段则由优化器自动生成索引。

（5）在条件表达式中经常用到的重复值较少的列上建立索引，避免在重复值较多的列上建立索引。有大量重复值并且经常有范围查询（BETWEEN，＞，＜，＞＝，＜＝）时可考虑建立聚簇索引。

（6）SELECT、UPDATE、DELETE语句中的子查询应当有规律地查找少于20％的表行。如果一个语句查找的行数超过总行数的20％，它将不能通过使用索引获得性能上的提升。

一般来说，当检索的数据超过20％时，数据库将选择全表扫描，而不使用索引。也就是说，表很小或者查询将检索表的大部分时，检索并不能提高性能。最好的情况是，将一些列

包含在索引中，而查询恰好包含由索引维护的那些行，此时优化器将从索引直接提供结果集，而不用回到表中去取数据。

(7) 如果建表时就建立索引，那么在输入初始数据时，每插入一条记录都要维护一次索引。系统在使用一段时间后，索引可能会失效或者因为频繁操作而使得读取效率降低，当系统效率降低或使用索引不明不白地慢下来的时候，可以使用工具检查索引的完整性，必要时进行修复。另外，当数据库表更新大量数据后，删除并重建索引可以提高查询速度。

(8) 如果表中对主键查询较少，并且很少按照范围检索，就不要将聚集索引建立在主键上。由于聚集索引每张表只有一个，因此应该根据实际情况确定将其分配给经常使用范围检查的属性列，这样可以最大限度地提高系统的运行效率。

(9) 比较窄的索引具有较高的效率。对于比较窄的索引来说，每页上能存放较多的索引行，而且索引的深度也比较少，所以，缓存中能放置更多的索引页，这样也减少了 I/O 操作。

(10) 不应该对包含大量 NULL 值的字段设置索引。就像代码和数据库结构在投入使用之前需要反复进行测试一样，索引也是如此。应该用一些时间来尝试不同的索引组合。索引的使用没有什么固定的规则，需要对表的关系、查询和事务需求、数据本身有透彻的了解才能最有效地使用索引。索引也不是越多越好，只有适度参照上面的原则使用索引才能取得较好的效果。

注意：表和索引都应该进行事先的规划，不要认为使用索引就能解决所有的性能问题，索引有可能根本不会改善性能(甚至可能降低性能)而只是占据磁盘空间。

在使用索引时可以有效地提高查询速度，但如果 SQL 语句使用得不恰当的话，所建立的索引就不能发挥作用。所以应该做到不但会写 SQL 语句，还要写出性能优良的 SQL 语句。

5.4.2 查询语句的优化

下面介绍的一些针对 SQL 语句的优化方法，虽然查询优化器已经帮用户做了很多优化，但是熟识这些方法可以在实际使用中提高查询效率。

1. 避免和简化排序

简化或避免对大型表进行重复的排序，会极大地提高 SQL 语句的执行效率。当能够利用索引自动以适当的次序产生输出时，优化器就避免了排序的步骤。

为了避免不必要的排序，必须正确地建立索引；如果排序不可避免，则应当简化它，如缩小排序列的范围等。在嵌套查询中，对表进行顺序存取对查询的效率可能产生致命的影响。如果采用顺序存取策略，一个嵌套 3 层的查询，如果每层都查询 1000 行，那么该查询就要查询 1 亿行数据，避免这种情况的主要方法就是对连接的列进行索引或使用并集来避免顺序存取。

【例 5.3】 OR 不能使用索引，可以用 UNION 来代替，以避免全表扫描。

```
SELECT Sno,Grade FROM SC
```

```
    WHERE (Sno = '2007111001' and Grade > = 60) or Grade > = 80
调整为 -->
SELECT Sno, Grade FROM SC WHERE Sno = '2007111001' and Grade > = 60
    UNION
SELECT Sno, Grade FROM SC WHERE Grade > = 80
```

2. 消除对大型表行数据的顺序存储(避免顺序存取)

在嵌套查询中,表的顺序存取对查询效率可能会产生致命的影响,比如一个嵌套3层的查询采用顺序存取策略,如果要每层都查询1000行,那么这个嵌套查询就要查询10亿行数据。避免这种情况的主要方法就是对连接列进行索引,还可以使用并集(UNION)来避免顺序存取。尽管在所有的检查列上都有索引,但某些形式的WHERE子句强迫优化器使用顺序存取。

3. 避免相关子查询

相关查询效率不高,如果在主查询和WHERE子句中的查询中出现了同一个列标签,就会使主查询的列值改变后,子查询也必须重新进行一次查询。查询的嵌套层次越多,查询的效率就会越低,所以应该避免子查询。如果子查询不可避免,那么就要在查询的过程中过滤掉尽可能多的行。

4. 避免困难的正规表达式

MATCHS和LIKE关键字支持通配符匹配,技术上叫做正规表达式,但这种匹配较为消耗时间,另外还要避免使用非开始的子串,如

```
SELECT * FROM employee WHERE name LIKE '% zhang'
```

写为

```
SELECT * FROM employee WHERE name LIKE 'z % '
```

更有效。

5. 使用临时表加速查询

对表的一个子集进行排序并创建临时表,也能实现查询加速。在一些情况下,这样做有助于避免多重排序操作,而且在其他方面还能简化优化器的工作。

【例5.4】 创建临时表加速查询。

```
SELECT S.Sno, Sname, SC.Cno, Grade FROM S, SC
WHERE S.Sno = SC.Sno AND SC.Grade >= 90 ORDER BY Same
INTO TEMP S_SC
```

用如下方式在临时表中进行查询:

```
SELECT * FROM S_SC WHERE Sname = '张三'
```

所创建的临时表的行要比主表的行少,其物理顺序就是所要求的顺序,这样就减少了磁盘I/O,降低了查询的工作量,提高了效率,而且临时表的创建并不会反映主表的修改。

6. 用排序来取代非顺序存储

磁盘存取臂的来回移动使得非顺序磁盘存取变成了最慢的操作。但是在 SQL 语句中这个情况被隐藏了，这样就使得我们在写应用程序时很容易写出进行了大量的非顺序页的查询代码，降低了查询速度，对于这个现象还没有很好的解决方法，只能依赖于数据库的排序能力来替代非顺序存取。

7. 避免大规模排序操作

大规模排序操作意味着使用 ORDER BY、GROUPE BY 和 HAVING 子句。无论何时执行排序操作，都意味着数据自己必须保存到内存或磁盘里(当以分配的内存空间不足时)。数据是经常需要排序的，排序的主要问题是会影响 SQL 语句的响应时间。由于大规模排序操作不是总能够避免的，所以最好把大规模排序在批处理过程里，在数据库使用的非繁忙期运行，从而避免影响大多数用户进程的性能。

8. 避免使用 IN 语句

当查询条件中有 IN 关键字时，优化器采用 OR 并列条件。数据库管理系统将对每一个 OR 从句进行查询，将所有的结果合并后去掉重复项作为最终结果，当可以使用 IN 或 EXIST 语句时应考虑如下原则：EXIST 远比 IN 的效率高，在操作中如果把所有的 IN 操作符子查询改写为使用 EXIST 的子查询，这样效率更高。同理，使用 NOT EXIST 代替 NOT IN 会使查询添加限制条件，由此减少全表扫描的次数，从而加快查询的速度以达到提高数据库运行效率的目的。

9. 使用 WHERE 代替 HAVING

HAVING 子句仅在聚集 GROUP BY 子句收集行之后才施加限制，这样会导致全表扫描后再选择，而如果可以使用 WHERE 子句来代替 HAVING，则在扫描表的同时就进行了选择，其查询效率大大提高了。但当 HAVING 子句用于聚集函数，不能有 WHERE 代替时则必须使用 HAVING。

10. 避免使用不兼容的数据类型

float 和 int，char 和 varchar，binary 和 varbinary 是不兼容的数据类型。数据类型的不兼容会使优化器无法执行一些本来可以进行优化的操作。例如：

```
SELECT * FROM SC WHERE Grade > 62.5
```

在这条语句中，如果 grade 字段是 int 型的，则优化器很难对其进行优化，因为 62.5 是个 float 型的数据。应该在编程时将浮点型转化为整型，而不是等到运行时再转化。

5.5 本章小结

查询处理是数据库管理的核心，而查询优化又是查询处理的关键技术。查询优化是任

何数据库系统都会面临的问题。对于关系数据库来说，由于其所依据的理论的特点，查询优化问题的研究与解决反倒成为其得以蓬勃发展的重要机遇。查询优化一般可分为代数优化和物理优化。代数优化是指关系代数表达式的优化；物理优化是指存储路径和底层操作算法的优化。本章在介绍了查询的处理过程后，重点讨论了查询的优化方法，以及代数优化和物理优化，最后指出在实际使用数据库的过程中应该注意能提高查询优化的方法，合理使用索引和查询语句的优化。

5.6 习题

5.6.1 简答题

1. 试述关系数据库查询处理的过程。
2. 试述查询优化的一般方法。
3. 试述使用索引的一般准则。
4. 试述查询语句优化的一般方法。

5.6.2 综合题

对学生课程数据库有如下查询：

```
SELECT Cno FROM S,C,SC WHERE S. Sno = SC.Sno AND SC.Cno = C.Cno AND S.Sdept = 'IS';
```

此查询要求给出信息系学生选修了的所有课程名称。

试画出用关系代数表示的语法树，并用关系代数表达式优化算法对原始的语法树进行优化处理，画出优化后的标准语法树。

第6章 数据库保护

数据库系统中的数据由 DBMS 统一控制和管理，从而保证数据库中数据的安全可靠和正确有效。数据库的保护主要涉及数据的安全性、完整性、并发控制和数据库恢复四个方面。

6.1 数据库安全性

数据库的安全性是指保护数据库，防止用户不合法使用所造成的数据泄露、修改或破坏。

6.1.1 数据库安全性概述

数据库一个最大的特点就是数据共享，但数据共享必然会带来一系列的数据库安全性问题。数据库中存放了大量的企业、组织或者个人的数据，其中许多数据可能是非常关键的、机密的或者涉及个人隐私，如国家机密、军事秘密、新产品实验数据、市场营销策略、市场销售计划、客户档案、银行储蓄数据等，通常要求只允许部分有权限的人才可以访问这些数据。如果 DBMS 不能严格地保证数据库中数据的安全性，就会严重制约数据库的应用。

因此，数据库系统中的数据共享不能是无条件的共享，而必须是在 DBMS 统一的严格控制和管理下，只允许有合法权限的用户访问允许他存取的数据，数据库管理系统通常采用种种措施防止用户越权使用数据库，数据库系统的安全保护措施是否有效是数据库系统主要的性能指标之一。

与数据安全性密切相关的是数据的保密问题，即合法用户合法地访问到机密数据后能否对这些数据保密，这在很大程度上属于法律、政策、伦理、道德方面的问题，而不属于技术上的问题，故在本书中不予讨论。事实上，一些国家已经成立了专门的机构对数据保密制定了相应的法律道德准则和政策法规。

6.1.2 数据库安全性策略

实际上，安全性问题并不是数据库系统所独有的，所有计算机系统中都存在这个问题，只是由于数据库系统中存放了大量数据，并为许多用户直接共享，因此使得其安全性问题更为突出。在一般的计算机系统中，安全措施往往是一级一级层层设置的。图 6-1 所示为一种很常用的安全模型。

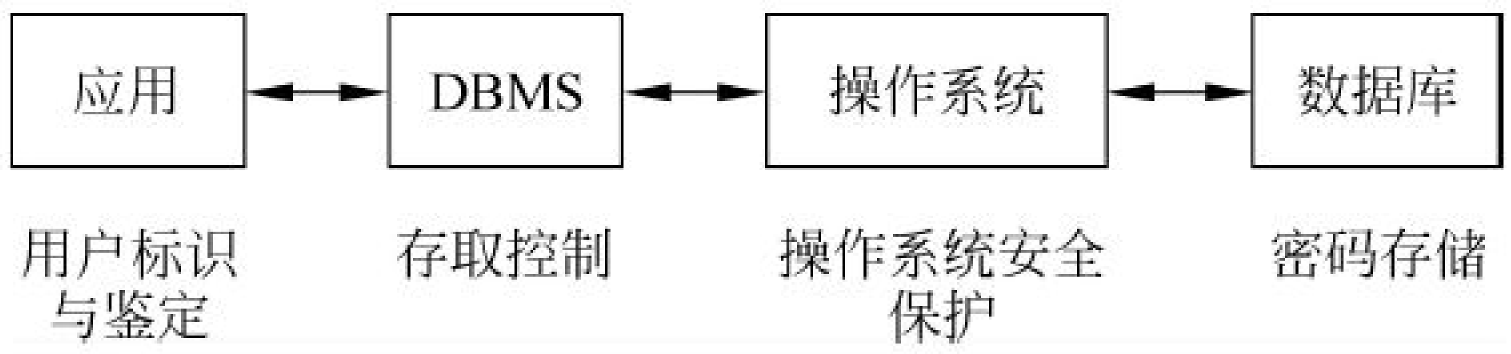

图 6-1 计算机系统的安全模型

在图 6-1 所示的安全模型中，当用户要求进入计算机系统时，系统首先根据用户输入的用户标识进行身份鉴定，只有合法的用户才允许进入计算机系统；对已经登录的用户，DBMS 还要对数据的存取权限加以控制，只允许用户执行合法操作；同时，操作系统也会有自己的保护措施，防止用户非法获取文件；最后数据还可以以密码形式存储到数据库中。这里只讨论与数据库有关的用户标识与鉴定、存取控制、视图、审计以及数据加密等安全技术。

这些数据保护措施都是由计算机系统设置的。除此之外，数据保护措施还包括加强保安工作，防止通信线路窃听，防止盗窃物理存储设备等，这方面的内容也不在本章讨论之列。

1. 用户标识与鉴定

DBMS 要求严格的用户身份鉴定。一个 DBMS 可能要求用户传递指定的通行字和时间日期检查，这一认证是在操作系统完成的认证之外另加的。DBMS 在操作系统之外作为一个应用程序被运行，这意味着它没有得到操作系统的可信赖路径，因此必须怀疑它收到的任何数据，包括用户认证。因此 DBMS 最好有自己的认证机制。

用户标识和鉴定是系统提供的最外层安全保护措施，其方法是由系统提供一定的方式让用户标识自己的名字或身份。系统内部记录着所有合法用户的标识，每次用户要求进入系统时，由系统将用户提供的身份标识与系统内部记录的合法用户标识进行核对，通过鉴定后才提供机器使用权。获得机器使用权的用户不一定具有数据库的使用权，数据库系统需要进行进一步的用户标识和鉴定，以拒绝没有数据库使用权的用户（非法用户）进行数据库数据的存取操作。

用户标识与鉴定的方法有很多种，常用的有以下三种。

- 用户的专门知识：如口令、密码和一组预定的问答等。
- 用户的特有东西：如徽章、磁卡、钥匙等。
- 用户的个人特征：如声音、指纹、签名等。

（1）用户的专门知识识别

使用只有用户自己知道的特征知识来识别用户是最常用的一种方法，一般采用口令或者密码，有时使用只有用户自己能给出正确答案的一组问题，有时还可以将两者结合起来使用。

使用这类方法要注意：

- 标识的有效性。口令、密码或者问题答案要尽可能准确地标识每一个用户。
- 内容的简易性。口令、密码要长短适中，问答过程不要太烦琐。
- 本身的安全性。为了防止口令、密码或问题答案的泄露或失窃，需要经常更改。

实现这种方法需要使用专门的软件来进行用户 ID 及其口令的登记、维护和检验等，但

它不需要额外专门的硬件设备，这是它的优点。其主要的缺点是口令、密码或者问题答案被泄密（包括无意或故意）给别人，不会有任何痕迹，不易被发觉，所以存在安全隐患。

（2）用户的特有东西识别

让每一个用户持有一个他所特有的物件，如徽章、磁卡、钥匙等。识别时，将其插入一个“阅读器”中，读取其面上的磁条信息。该方法是目前一些安全系统中比较常用的一种方法，但用在数据库系统中需要考虑以下两点：

- 需要专门的阅读装置。
- 要求从阅读器中抽取信息及与 DBMS 接口的软件。

该方法的优点是比个人特征识别更简单、有效，性价比更高。其缺点是容易忘记带徽章、磁卡或钥匙等，也有可能丢失甚至被人盗取。有时在无这种特有的物件情况下，用户为了及时完成他的任务，就临时采用替代的办法，而这本身又会危及系统安全。

（3）用户的个人特征识别

使用每个人所具有的个人特征，如声音、指纹、签名等来识别用户，是当前最有效的方法。但是有两个问题必须解决：

- 专门设备。要能准确地记录、存储和存取这些个人特征。
- 识别算法。要能较准确地识别出每个人的声音、指纹或签名。这里的关键问题是“有效性测度”，要让“合法者被拒绝”和“非法者被接受”的误判率达到实用的程度，或者达到应用环境可接受的程度。百分之百正确，即误判率为零几乎是不可能的。另外，代价也是不得不考虑的问题，这不仅是经济上的代价，还包括识别算法执行的时空代价，它影响着整个安全子系统的性能比。

2. 存取控制

在数据库中，为了保证用户只能访问他有权存取的数据，必须预先对每个用户定义存取权限。对于通过鉴定进入系统的用户（即合法用户），系统根据他的存取权限定义对他的各种操作请求进行控制，确保他只执行合法操作，而对于未授权的用户（非法用户）则无法接近数据，这主要由数据库系统的存取控制机制来完成。

（1）存取控制机制的构成

存取控制机制主要包括两部分：

① 定义用户权限，并将用户权限登记到数据字典中。

用户权限是指用户对于数据对象能够执行的操作种类。要进行用户权限定义，系统必须提供有关定义用户权限的语言，该语言称为数据控制语言 DCL（Data Control Language）；具有授权资格的用户使用 DCL 描述授权决定，并将授权决定告知计算机；计算机分析授权决定，并将编译后的授权决定存放在数据字典中。

② 当用户提出操作请求时，系统进行权限检查，拒绝用户的非法操作。

每当用户发出存取数据库的操作请求后，DBMS 首先查找数据字典，进行合法权限检查。如果用户的操作请求没有超出其数据操作权限，则准予执行该数据操作；否则系统将拒绝执行此操作。

（2）存取控制机制的类别

当前，大多数 DBMS 所采用的存取控制策略主要有两种：自主存取控制和强制存取控

制。其中，自主存取控制的使用更为普遍，也有些系统两者都支持。下面分别介绍这两种方法。

① 自主存取控制

在自主存取控制(Discretionary Access Control，DAC)方法中，用户对于不同的对象有不同的存取权限，不同的用户对同一对象的存取权限也各不相同，用户可将自己拥有的存取权限转授给其他用户。很明显，自主存取控制比较灵活。

② 强制存取控制

在强制存取控制(Mandatory Access Control，MAC)方法中，每一个数据对象被标以一定的密级，每一个用户也被授予某一个级别的许可证。对于任何一个对象，只有具有合法许可证的用户才可以存取。与自主存取控制相比，强制存取控制比较严格。

(3) 自主存取控制方法

SQL 对自主存取控制提供了支持，其 DCL 主要是通过 GRANT(授权)语句和 REVOKE(收权)语句来实现的。

① 关系数据库中的存取权限

存取权限是由数据对象和操作类型组成的。在数据库系统中，定义用户的存取权限称为授权，即规定可以对数据库中的哪些数据对象进行哪些操作。如果用户的操作超出了系统对他的授权，则该操作将不会被执行。

关系数据库系统的存取权限如表 6-1 所示。

表 6-1 关系数据库系统的存取权限

数据对象		操作类型
模式	外模式	建立、修改、检索
	模式	建立、修改、检索
	内模式	建立、修改、检索
数据	表	查找、插入、修改、删除
	属性	查找、插入、修改、删除

② SQL 的数据控制功能

SQL 的数据控制功能包括 GRANT(授权)语句和 REVOKE(收权)语句。

- 数据对象的创建者自动获得对于该数据对象的所有操作权限。例如，学生表(STUDENT)的创建者自动获得对该表的 SELECT、INSERT、UPDATE 和 DELETE 等权限。
- 获得数据操作权的用户可以通过 GRANT 语句把权限转授给其他用户。

GRANT 语句的一般格式为：

```
GRANT <权限> [,<权限>]…
ON <对象类型> <对象名>[,<对象类型> <对象名>]…
TO <用户> [,<用户> ]
[WITH GRANT OPTION]
```

例如：

```
GRANT SELECT,INSERT
```

```
ON STUDENT  TO 张鹏
WITH GRANT OPTION
```

执行结果是将学生表的 SELECT 和 INSERT 权限授予给了用户张鹏，同时张鹏还获得了“授权”权限，即可以把得到的权限继续授予其他用户。

- 当用户将某些权限授给其他用户后，有时还需要把权限收回。收回权限使用 REVOKE 语句。

REVOKE 语句的一般格式为：

```
REVOKE <权限> [,<权限>]…
ON <对象类型> <对象名>[,<对象类型> <对象名>]…
FROM <用户> [,<用户> ]
[CASCADE|RESTRICT]
```

例如：

```
REVOKE INSERT ON STUDENT FROM 张鹏
```

执行结果是收回张鹏对 STUDENT 表的插入权限。

③ 授权机制的性能

衡量授权机制灵活性的一个重要指标就是授权粒度，即可定义的数据对象范围。授权粒度越细，可定义的数据对象范围越小，授权机制越灵活。

④ 自主存取控制的不足之处

自主存取控制能够通过授权机制有效地控制用户对敏感数据的存取，但也存在着一定的缺陷，其主要问题是系统无法对授权的授予状况进行有效的控制，因此有可能造成数据的无意泄露。例如，甲将自己权限范围内的某些数据存取权限授权给乙，甲的本意是只允许乙本人操作这些数据。但是甲的这种安全性要求并不能够得以保证，因为乙一旦获得数据的权限，就可以把数据备份，获得自身权限内的副本，在不征得甲同意的前提下进行副本传播，从而造成数据泄露。造成这个问题的根本原因在于，自主存取控制机制仅仅通过对数据的存取权限进行安全性控制，而数据本身并没有安全性标记。强制存取控制方法则可以有效解决这一问题。

(4) 强制存取控制方法

一般情况下，自主存取控制是很有效的，可以满足普通的安全性要求。但它存在一个漏洞：一些别有用心的用户可以欺骗一个授权用户，采用一定的手段来获取敏感数据。存在这种漏洞的根源在于，自主存取控制机制仅以授权来将主体(用户)与客体(被存取数据对象)关联，通过控制权限来实现安全要求，对主体和客体(对象)本身未做任何安全性标注。强制存取控制(MAC)能够处理自主存取控制的这种漏洞问题。

通用的强制存取控制机制将被存取的物体称为客体或对象(Object)，如文件、关系、模式、视图、索引、记录/元组等；将存取客体者称为主体(Subject)，包括用户和程序；它给每一个客体指派一个安全等级(Security class)，而给每一个主体指派一个安全等级的许可证(Clearance)。安全级别(主体和客体的统一)按偏序组织，例如绝密(top secret，TS)＞机密(secret，S)＞秘密(confidential，C)＞无密(unclassified，U)，这里“＞”表示安全级别的“高于”关系，它是一个偏序关系。

强制存取控制机制使用两条安全性规则：

① 主体 A 可以读取客体 O 的必要充分条件是 CL(A)≥SC(O)

② 主体 A 可以更新客体 O 的必要充分条件是 CL(A)＝SC(O)

其中，CL(A)和 SC(O)分别表示主体 A 的许可证等级和客体 O 的安全等级。规则①的意义很明确，规则②可以用另一种形式表述：一个主体所写的对象的密级自动为主体的许可证级别。这两个规则控制了比对象的密级更高的用户只能读而不能更新该对象；密级更低的用户既不能读也不能更新；只有同级的用户才既能读又能更新。这也就防止了自主存取控制关于“复制”的漏洞。

强制存取控制方案的优点是它更严密，而其主要缺点就在于它的“刚性”太强，其安全策略必须由 DBA 设置，而且在安全分级机制等方面也不够灵活。将自主和强制存取控制组合起来效果会更理想些，可以在强制存取控制下再施加自主存取控制。这样，一个用户要存取一个数据库对象，他必须有相应的存取权限(经授权获得)，且他的许可证等级与该对象的密级必须受到相关规则的约束。

实现 MAC 时首先要实现 DAC，即 DAC 与 MAC 共同构成 DBMS 的安全机制，如图 6-2 所示。系统在进行安全检查时，首先进行自主存取控制检查，然后进行强制存取控制检查，两者都通过后，用户才能执行其数据存取操作。

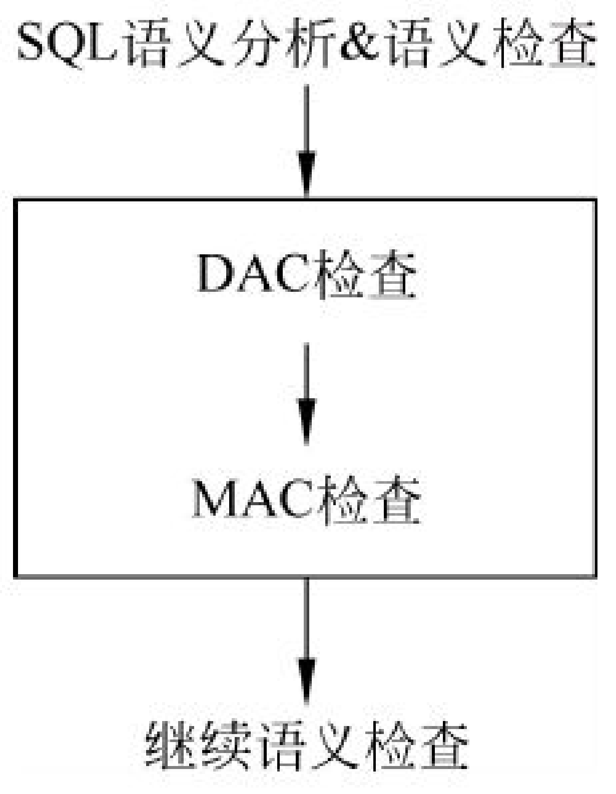

图 6-2 DAC＋MAC 安全性检查

3. 视图机制

进行存取的控制，不仅可以通过授权与收回权力来实现，还可以通过定义用户的外模式来提供一定的安全保护功能。在关系系统中，就是为不同的用户定义不同的视图，通过视图机制把要保密的数据对无权存取这些数据的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

4. 审计

用户识别和鉴定、存取控制、视图等安全性措施均为强制性机制，将用户操作限制在规定的安全范围内。但实际上任何系统的安全性措施都不可能是完美无缺的，蓄意盗窃、破坏数据的人总会想方设法打破控制。所以，当数据相当敏感，或者对数据的处理极为重要时，就必须以审计追踪技术作为预防手段，监测可能发生的不合法行为。

审计追踪使用的是一个专用文件或数据库，系统自动将用户对数据库的所有操作记录在上面，利用审计追踪的信息，就能重现导致数据库现有状况的一系列事件，以找出非法存取数据的人、时间和内容等。

审计追踪通常是很费时间和空间的，所以一般 DBMS 把审计功能作为可选特性，DBA 可以根据需要灵活地打开或者关闭审计功能。审计功能一般用于对安全性要求较高的部门。

5. 数据加密

前面所述的数据库安全措施都是在 DBMS 的管理控制下起作用的，现在的问题是有人

会有意或者无意地绕过或者避开 DBMS 而窃取数据。所以对于高度敏感性数据，如财务数据、军事数据和国家机密，除了采取上述安全性措施外，还可以采用数据加密技术，以密码形式存储和传输数据。加密的目的是使未授权的人不能识别，也不知道如何识别，这样就无法企图通过不正常渠道获取数据，例如，利用系统安全措施的漏洞非法访问数据，或者在通信线路上窃取数据，而只能看到一些无法辨认的二进制代码。用户正常检索数据时，首先要提供密码钥匙，由系统进行译码后才能得到可识别的数据，也就是将密文转变为明文，这个过程称为解密，是加密的逆过程。

目前不少数据库产品都提供了数据加密例行程序，可根据用户的要求自动对存储和传输的数据进行加密处理。另外一些数据库产品虽然本身未提供加密程序，但是提供了接口，允许用户用其他厂商的机密程序对数据加密。所有提供加密机制的系统必然也提供相应的解密程序。这些解密程序本身也必须具有一定的安全性保护措施，否则数据加密的优点也就遗失殆尽了。

数据加密和解密程序也是比较费时的，而且数据加密和解密程序会占用大量的系统资源，所以数据加密功能通常也作为可选特性，用户可以根据需要自由选择只对高度机密的数据加密。

6.2 数据库完整性

数据库的完整性是指保证数据库数据的正确性和相容性，防止错误的数据进入数据库。数据库是否具备完整性关系到数据库系统能否真实地反映现实世界，因此维护数据库的完整性是非常重要的。

6.2.1 完整性概述

数据库的完整性和安全性是两个不同的概念。前者是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出，即所谓垃圾进垃圾出所造成的无效操作和错误结果；而后者是保护数据库，防止恶意的破坏和非法的存取。也就是说，安全性措施的防范对象是非法用户和非法操作，完整性措施的防范对象是不合语义的数据。当然，完整性和安全性是密切相关的。安全性可以用用户对数据库的操作权限来表示，完整性可以用完整性约束来表示。用户操作权限和完整性约束都存储在数据库管理系统的数据字典中。数据库的安全性和完整性都由 DBMS 来确保。

为维护数据库的完整性，DBMS 必须提供一种机制来检查数据库中的数据，看其是否满足语义规定的条件，这些语义约束条件称为数据库完整性约束条件。数据库中的完整性约束条件实际上是数据语义的表示形式，是由用户定义的。大部分约束条件可在定义数据模式时定义，成为数据模式的一部分。数据库完整性约束的种类很多，可以根据数据的实际语义和用户的业务规则或政策定义数据上的完整性约束。

目前，许多 RDBMS 提供了多种定义完整性约束条件的功能和检查是否违背完整性约束条件的机制，称为完整性检查。RDBMS 若发现用户的操作使数据库违背了完整性约束条件，将采取一定的措施拒绝用户执行该操作。数据完整性检查一般设置在引起数据库状

态改变的操作上,不引起数据库状态改变的查询等操作不会破坏数据库的完整性。

数据库的完整性规则主要包括实体完整性、参照完整性、用户定义完整性。除此之外,还有域完整性。域完整性也可以称为列完整性,用于指定一个数据集对某一个列是否有效和确定,是否允许空值。域完整性通常是使用有效性检查来实现的,并且还可以通过限制数据类型、格式或者可能的取值范围来实现。例如,在性别列中,限制其取值范围为“男”和“女”,这样该列就不会输入其他一些无效的值。

6.2.2 完整性约束条件

完整性控制都是围绕完整性约束条件进行的,从这个角度说,完整性约束条件是完整性控制机制的核心。

完整性约束条件作用的对象可以有列级、元组级和关系级三种粒度。其中对列的约束主要指对其取值类型、范围、精度和排序等的约束条件。对元组的约束是指对记录中各个字段间联系的约束。对关系的约束是指对各记录间或关系之间联系的约束。

完整性约束条件设计的这三类对象,其状态可以是静态的,也可以是动态的。其中对静态对象的约束是反映数据库状态合理性的约束,这是最重要的一类完整性约束。对动态对象的约束是反映数据库状态变迁的约束。完整性约束条件可以分为六类。

1. 静态列级约束

静态列级约束是对一个列的取值域的说明,这是最常见、最容易实现的一类完整性约束,包括以下几个方面。

(1) 对数据类型的约束

包括数据的类型、长度、单位和精度等。例如,学生-课程数据库中学生姓名的数据类型为字符型,长度为8。教师数据库中教师的工资类型为货币型。

(2) 对数据格式的约束

例如,规定职工编号前两位为参加工作年份,中间两位为部门编号,后四位为顺序编号。

(3) 对取值范围或取值集合的约束

例如,规定职工性别只能取自集合[男,女],职工年龄只能是20～55岁。

(4)对空值的约束

空值表示未知或未赋值,与零值和空字符不同。有的列可以取空值,有的不可以。例如,职工编号不能取空值,联系电话不能取空值。

(5) 其他约束

例如列排序说明、列组合等。

2. 静态元组约束

一个元组是由若干个列值组成的,静态元组约束就是规定组成一个元组的各个列之间的约束关系。

静态元组约束只局限在单个元组上,因此比较容易实现。例如,在图书借阅表中可以规定:还书日期>借书日期。

3. 静态关系约束

在一个关系的各个元组之间或者若干关系之间常常存在各种联系或约束。常见的静态关系约束有以下四种：

(1) 实体完整性约束。

(2) 参照完整性约束。

(3) 函数依赖约束。

(4) 统计约束,即某个字段值与一个关系多个元组的统计值之间的约束关系。例如,规定部门经理的工资不得高于本部门职工平均工资的 4 倍,不得低于本部门职工平均工资的 2 倍,本部门职工的平均工资就是一个统计值。

4. 动态列级约束

动态列级约束是修改列定义或列值时要满足的约束条件,包括以下两方面。

(1) 修改列定义时的约束

例如,将原来允许为空值的列改为不可以为空值,如果之前已经取空值,则拒绝本次修改。

(2) 修改列值时的约束

修改列值时需要参考该列之前的旧值,并且新旧值之间满足某种约束关系。例如,职工新工资不能低于原来的工资,学生年龄只能增长而不能减少。

5. 动态元组约束

动态元组约束是指修改某个元组时需要参照其旧值,并且新旧值之间需要满足某种约束条件。例如,教师表中有职称和工资字段,规定调整工资时教授的工资不得低于 5000 元。

6. 动态关系约束

动态关系约束是加在关系变化前后状态上的限制条件,例如事务一致性、原子性等约束条件。

6.2.3 完整性控制

1. 完整性约束的定义

DBMS 的完整性控制机制应具有三个方面的功能：

(1) 定义功能：提供定义完整性约束条件的机制。

(2) 检查功能：检查用户发出的操作请求是否违背了完整性约束条件。

(3) 如果发现用户的操作请求使数据违背了完整性约束条件,则采取一定的动作来保证数据的完整性。

2. 对违背了完整性约束条件的操作应采取的措施

对于违反实体完整性规则和用户定义的完整性规则的操作一般都是采用拒绝执行的方

式进行处理。而对于违反参照完整性的操作，并不都是简单地拒绝执行，有时还需要采取另一种方法，即接受这个操作，同时执行一些附加的操作，以保证数据库的状态仍然是正确的。

(1) 删除被参照关系的元组时的考虑

① 级联删除(Cascades)。将参照关系中所有外码值与被参照关系中要删除元组主码值相对应的元组一并删除。如果参照关系同时又是另一个关系的被参照关系，则这种删除会继续级联下去。例如，要删除学生表 Student 中“学号='2007111001'”的学生，则一起删除选课表 *SC* 中所有“学号='2007111001'”的元组。

② 受限删除(Restricted)。只有当参照关系中没有任何元组的外码值与要删除的被参照关系的主码值相对应时，系统才执行删除操作，否则拒绝执行。例如，如果 *SC* 表中有“学号='2007111001'”的学生的选课记录，则上面删除学生表 Student 中“学号='2007111001'”的学生的操作将被拒绝执行。

③ 删除值空置(Nullifies)。删除被参照关系中的元组，并将参照关系中所有与被参照关系中删除元组主码值相等的外码值置为空值。例如，上面将 *SC* 表中的所有“学号='2007111001'”的元组的学号置为空值。

对于这三种处理方法，要根据应用环境的语义来确定采用哪一种最为恰当。例如，一个学生毕业或者退学了，他的记录就应该从 Student 表中删除，同样，他的选课记录也应该从 *SC* 表中一起全部删除。

(2) 修改被参照关系中主码的考虑

① 级联修改(Cascades)。修改被参照关系中的主码值的同时修改参照关系中与之对应的外码值。如，学生表 Student 中“学号='2007111001'”的元组修改为“学号='2007111002'”，则同时将选课表 *SC* 中所有“学号='2007111001'”的元组修改为“学号='2007111002'”。

② 受限修改(Restricted)。拒绝此修改操作。只有当参照关系中没有任何元组的外码值与被参照关系中某一个元组的主码值相等时，才允许此修改操作，否则拒绝执行。例如，只有当 *SC* 表中没有“学号='2007111001'”的元组时，才允许对学生表 Student 将“学号='2007111001'”的元组修改为“学号='2007111002'”。

③ 修改值空置(Nullifies)。修改被参照关系中的主码值，同时把参照关系中相应的外码值置为空值。例如，把 Student 表中的“学号='2007111001'”的元组修改为“学号='2007111002'”，同时把 *SC* 表中“学号='2007111002'”的元组的学号置为空值。

对于这三种处理方法，同样要根据应用环境的语义来确定采用哪一种最为恰当。

(3) 外码是否可以接受空值

外码是否可以接受空值是由其语义来决定的。在职工-部门数据库中，“职工”关系包含有外码“部门号”，某一元组的这一列若为空值，表示这一职工尚未分配到任何具体的部门工作；这和应用环境的语义是相符的，因此“职工”表的“部门号”列应允许空值。但在学生-选课数据库中，“学生”关系为被参照关系，其主码为“学号”；“选课”关系为参照关系，外码为“学号”，如果学号为空值，则无法表示何人选修了某门课程，这和应用环境的语义是不相符的，因此“选课”表的“学号”列不能为空值。

从上面的讨论可以看出，DBMS 在实现参照完整性时，除了要提供定义主码、外码机制外，还需要提供不同的策略供用户选择。无论哪一种策略，都应该根据应用环境的具体要求

来确定,选择一种最为适当的处理方式。

6.3 数据库并发控制

数据库是一个可以供多个用户共享的信息资源。当多个用户并行地存取数据库时,就会产生多个事务同时存取同一数据的情况。如果对并发操作不加以控制就可能会出现存取不正确数据的情况,进而破坏数据库的一致性。因此,DBMS 必须提供并发控制机制。并发控制机制的好坏是衡量数据库管理系统性能的重要指标之一。

6.3.1 事务概述

1. 事务的定义

事务是数据库的基本逻辑工作单位,它包括用户定义的一系列操作,这些操作要么全做要么全不做,是一个不可分割的基本单位。一个事务可以是一条 SQL 语句、一组 SQL 语句或一段程序。事务和程序是两个概念,一般地,一个程序中包含多个事务。事务的开始与结束可以由用户显式地定义,也可以由系统按缺省规定自动划分。

数据库的主要责任是保存信息,因此它需要向用户提供保存当前程序状态的方法。同样,当事务执行过程中发生错误时,需要有一种方法使数据库忽略当前的状态,并回到前面保存的程序状态。这两种情况在数据库用语中分别称为提交事务和回滚事务。在 SQL 语言中,定义事务的语句有三条:

```
BEGIN TRANSACTION;
COMMIT;
ROLLBACK;
```

通常,事务以 BEGIN TRANSACTION 开始,以 COMMIT 或者 ROLLBACK 结束。COMMIT 表示事务的提交,即提交事务的所有操作,事务提交是将事务中所有对数据的更新写回到磁盘上的物理数据库中,事务正常结束;ROLLBACK 表示事务的回滚,即事务在运行过程中发生某种故障,事务无法继续执行下去,系统将事务中对数据库的所有更新操作全部撤销,回滚到执行事务前的状态。

2. 事务的特性

事务具备四个特性,即原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)和持久性(Durability),简称为 ACID 特性。

(1) 原子性

原子性就是事务执行的原子性。事务是一个不可分割的工作单元,事务中包括的诸操作要么全部执行,要么全部不执行。

(2) 一致性

事务的一致性指的是一个事务执行完成,数据库从一个一致性的状态转换到另一个一致性状态,即它是事务在其两个端点处必须保证数据库的正确性的一种限制。例如,某公司在银行有 A、B 两个账户,两个账户之间进行转账,从 A 账户取出 1000 元存入 B 账户。该

事务里包含两个操作：一个操作是从账户 A 中减去 1000 元，另一个操作是向账户 B 存入 1000 元。这两个操作要么全做，要么全不做，这样都会使数据库处于一致性状态。如果只做其中一个操作，那么用户逻辑上会发生错误，少了 1000 元，这时数据库就处于不一致状态。可见一致性和原子性密切相关。

(3) 隔离性

原子性和一致性只能保证单个事务夭折或成功完成时数据库的正确状态。当多个事务并发执行时，因其互相干扰可能会导致数据库的最终状态是不正确的。因此 DBMS 必须对它们的执行给予一定的控制，使若干并发执行的结果等价于它们一个接一个地串行执行的结果。也就是说，事务在执行过程中是互不干扰的、相互隔离的。

(4) 持久性

持久性是指一个事务一经提交，它的影响永久性地产生在系统中，也就是说其修改操作写到了数据库中。这种特性也称为永久性。

事务机制的这些属性保证了数据的一致性，确保了在系统失败时的可恢复性。

6.3.2 并发控制概述

并发控制是指在多用户的环境下，对数据库并发操作进行规范的机制。其目的是避免对数据的丢失修改、读脏数据与不可重复读等，从而保证数据的正确性与一致性。

并发控制在多用户模式下是十分重要的，但这一点经常被一些数据库开发人员忽视，而且因为并发控制的层次和类型非常丰富，有时使人们在选择时比较迷惑，不清楚衡量并发控制层次选择的原则和途径。

1. 并发操作可能产生的问题

为了更好地理解并发控制的概念，先看一个常见的并发操作的例子。

银行数据库中有一个账户表，其主要信息如表 6-2 所示。

表 6-2 Account(账户)表主要信息

属性名称	属性含义	属性类型
Id(键值列)	账户号	Char(10)
Uname	户主	Char(10)
Mdeposit	存入金额	Currency
Mpayout	支出金额	Currency
Mbalance	存款余额	Currency

某户主代表在银行 A 前台取款 2000 元，银行 A 出纳查询用户的存款信息显示银行存款余额 20 000 元，正在这时，银行 B 账户转账支票支付该账户 5000 元，机器查询也得到当前用户存款 20 000 元，这时银行 A 的出纳员看到用户存款超过了取款额，就支付了客户 2000 元并将用户存款改为 18 000 元，而银行 B 的操作员根据支票，将汇入的 5000 元加上，把用户的余额改为 25 000 元。很明显，银行将会损失 2000 元，因为银行 A 的出纳员所做的修改被覆盖了。

这是由于对并发操作控制失败造成的，由于没有对两个并发操作进行合理的隔离，并对

数据进行合理的锁定，导致经出纳员查询所得到的客户端数据集与数据库的数据不一致，结果便产生了丢失修改。

当多个用户访问相同的数据时，可能会出现三种问题。

（1）丢失修改（Lost Update）：如果一个事务通过某种查询获取了数据，另一个事务修改了该数据的一部分，那么原来的事务第二次获取该数据时，就会发生虚读，产生丢失修改。上面的例子中，当银行 A 执行取款时，此时银行 B 已修改余额，可是银行 A 获取的数据仍为原数据，从而产生丢失修改。

（2）脏读（Dirty Read）：如果一个应用程序使用了被另一个应用程序修改过的数据，而这个数据处于未提交状态，这时就会发生脏读。第二个应用程序随后会请求回滚被其修改的数据，从而导致第一个事务使用的数据被损坏，即所谓的“变脏”。在上面的例子中，银行 B 修改余额后，银行 A 再次获取的数据为新数据，可是由于某些问题后来银行 B 回滚事务，取消了对余额的修改，则银行 A 获得了“脏数据”，即不正确的数据。

（3）不可重复读（Non-Repeatable Read）：如果一个事务获得了数据，而该数据随后被另一个事务所更改，那么第一个事务无法再现前一次读取的数据，称为不可重复读。在上例中，银行 B 修改余额后，如果银行 A 只能获取修改后的数据，就产生了不可重复读。

产生上述问题的原因是破坏了事务的隔离性。并发控制就是要用正确的方法调度并发操作，使一个事务的执行不受另一个事务的干扰，以避免造成数据的不一致。

2. 并发操作的调度

计算机系统对并行操作的调度是随机的，而不同的调度会产生不同的结果。如果一个事务执行时，不允许其他事务并发操作，即把事务串行地执行，则不会发生数据不一致的问题。因此，可以得到结论：当且仅当调度的结果与串行执行这些并发事务的结果相同时，这几个并发事务的执行才是正确的。这种并行调度策略称为可串行化的调度。

目前，大多数 DBMS 都采用封锁机制来进行并发控制。

6.3.3 封锁

封锁是指使事务对它要操作的数据对象有一定的控制能力，是并发控制的重要手段。封锁有三个步骤：第一步是申请加锁，即事务在操作前要对它使用的数据对象提出加锁请求；第二步是获得锁，即在条件成熟时，系统允许事务对数据加锁，从而使事务获得数据的控制权；第三步是释放锁，即完成操作后事务放弃对数据的控制权。为了达到封锁的目的，在使用时事务应该选择合适的锁，并且要遵从一定的封锁协议。

1. 封锁类型

（1）排他锁

排他锁（Exclusive Lock，简称 X 锁）又称写锁，是为修改数据而保留的。如果事务 T 对数据对象 A 加了 X 锁，则其他事务就不能再对 A 加任何类型的锁，直至事务 T 释放 A 上的 X 锁为止。这样，加了 X 锁的数据对象 A，其他事务不能读取也不能修改。排他锁是独占的。

（2）共享锁

共享锁（Share Lock，简称 S 锁）又称读锁，用于所有的只读数据操作。如果事务 T 对数

据对象A加了S锁，则其他事务也可以对A加S锁，但不能加X锁，直至释放A上的所有S锁为止。共享锁是非独占的，允许多个并发事务读取其锁定的数据资源。

2. 封锁粒度

封锁粒度是指封锁对象的大小。封锁对象可以是逻辑单元，也可以是物理单元。封锁的粒度越大，系统中能够被封锁的对象就越少，并发度也就越低，系统开销也越小；相反，封锁的粒度越小，系统中能够被封锁的对象就越多，并发度就越高，系统开销也越大。

选择封锁粒度要权衡系统开销和并发度。如果经常发生的并发操作是以关系为单位的，则封锁粒度可以为关系；如果经常发生的并发操作是几个关系，则封锁粒度可以为数据库。

3. 封锁协议

封锁的目的是能够正确地调度并发操作。为此，在运用X锁和S锁这两种基本封锁对一定粒度的数据对象加锁时，还需要约定一些规则，例如何时加锁、持续时间、何时释放等，一般称这些规则为封锁协议。对封锁方式规定不同的规则，就形成各种不同的封锁协议，它们分别在不同的程度上为并发操作的调度提供一定的保证。这里主要介绍保证数据一致性的三级封锁协议和保证并行调度可串行化的两段锁协议。

(1) 保证数据一致性的三级封锁协议

1级封锁协议是指事务在修改数据之前必须先对其加X锁，直到事务结束才释放。1级封锁协议可防止“虚读”产生的丢失修改，并保证事务T是可恢复的。

2级封锁协议是指在1级封锁协议的基础上，当事务在读取数据之前必须先对其加S锁，读完后即可释放S锁。2级封锁协议还可以进一步防止“脏读”。

3级封锁协议是指在1级封锁协议的基础上，当事务T在读取数据之前必须先对其加S锁，直到事务结束才释放。3级封锁协议还可以进一步防止“不可重复读”。

(2) 保证并行调度可串行化的两段锁协议

可串行性是并行调度正确的唯一准则，两段锁(Two-phase Locking，2PL)协议是保证并行调度可串行性的封锁协议。

两段锁协议规定，在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁；而且在释放一个封锁之后，事务不再获得任何其他封锁。

“两段”锁的含义是指事务分为两个阶段：获得封锁和释放封锁。

需要注意的是，事务的两段锁协议是可串行化调度的充分条件，而不是必要条件。也就是说，只要遵守两段锁协议的调度，就一定是可串行化的调度；反之，在可串行化的调度中，未必所有事务都遵守两段锁协议。

6.3.4 活锁与死锁

1. 活锁

假设事务T1封锁了某数据对象R，事务T2也申请封锁R，此时T2处于等待状态；可是当事务T1释放R后，系统允许新来的事务T3封锁了R，T2仍然处于等待状态；当事务

T3 释放 R 后，系统又允许新来的事务 T4 封锁 R，T2 还是处于等待状态；……。事务 T2 有可能一直等待下去永远无法执行，这就是活锁。试想一个秘书需要录入一封信，但她一直在忙于接电话，所以这封信永远都不会被录入，如图 6-3 所示。

T1	T2	T3	T4
Lock R	…	…	…
…	Lock R	…	…
…	等待	Lock R	…
Unlock R	等待	…	Lock R
…	等待	Lock R	等待
…	等待	…	等待
…	等待	UnLock R	等待
…	等待	…	Lock R
…	等待	…	

图 6-3　活锁

解决活锁的方法就是采用先来先服务的策略。当有多个事务请求封锁同一个数据对象 R 时，系统按照申请数据对象 R 的先后顺序排队，数据 R 上的锁一旦释放就允许申请队列中第一个事务封锁 R，释放封锁后将其从队列中删除。

2. 死锁

如果事务 T1 封锁了数据对象 R1，事务 T2 封锁了数据对象 R2；然后事务 T1 又申请封锁 R2，因为 R2 已被 T2 封锁，所以 T1 等待 T2 释放 R2 上的锁；接着事务 T2 又申请封锁 R1，因为 R1 已被 T1 封锁，所以 T2 等待 T1 释放 R1 上的锁。这样就出现了事务 T1、T2 互相等待的问题，T1、T2 永远无法结束，这就产生了死锁，有些类似于操作系统中的线程死锁问题，如图 6-4 所示。

T1	T2
Lock R_1	…
…	Lock R_2
…	…
Lock R_2	…
等待	…
等待	Lock R_1
等待	等待
等待	等待
…	…

图 6-4　死锁

数据库中解决死锁有两类方法：其一是预防死锁的发生，即采用预防协议使死锁状态永远都无法形成；其二是允许死锁发生，再采用检测和解决机制进行恢复。如果死锁发生的概率较高，则适合采用预防策略，否则采用死锁检测和恢复机制更为有效。

(1)死锁的预防

预防死锁通常有两种方法。

① 一次封锁法

一次封锁法要求每个事务必须一次将所有要使用的数据对象全部封锁，否则就不能执行。事务 T1 同时封锁了 R1 和 R2，执行结束后释放 R1 和 R2；然后事务 T2 再同时封锁了 R1 和 R2，直到其执行结束才释放 R1 和 R2。

一次封锁法能有效地防止死锁的发生，但也存在问题。其一是要一次性将以后用到的全部数据加锁，势必扩大了封锁的范围，从而降低了系统的并发度；其二是数据库中的数据是不断变化的，原来不需要加锁的数据，在执行过程中可能会变成封锁对象，所以很难事先

精确地确定每个事物要封锁的全部数据对象，只能扩大封锁范围，将事务在运行过程中可能要封锁的数据对象全部加锁，这就进一步降低了并发度。

② 顺序封锁法

顺序封锁法是预先对数据对象规定一个封锁顺序，所有事务都按这个顺序封锁数据对象。假设数据对象的封锁顺序为 R1→R2，则事务 T1、事务 T2 都必须先封锁 R1 再封锁 R2。

顺序封锁法也能够有效地防止死锁的发生，但是维护数据对象的封锁顺序是很麻烦的事情。因为数据库中的数据是不断动态变化的，而且事务的封锁请求可以随着事务的执行而动态地决定，有时很难按照既定的顺序进行封锁。

由此可见，在操作系统中普遍采用的死锁预防策略并不是很适合数据库系统的特点，因此，DBMS 解决死锁问题普遍采用的是死锁诊断与解除的方法。

(2) 死锁的诊断与解除

诊断死锁需要使用事务等待图，它动态地反映所有事务的等待情况，并发控制子系统周期性地检测事务等待图，只要在图中出现回路，就说明存在死锁。

一旦出现死锁就要设法解除，通常选择处理代价最小的事务，将其回滚，释放所有它持有的封锁，使其他事务能继续执行下去。

6.4 数据库恢复

6.4.1 数据库恢复概述

在数据库系统中，恢复的基本含义是恢复数据库本身，即在发生某种故障使数据库数据不再正确时，把数据库恢复到已知正确的某一状态。

数据库故障是指数据库运行过程中影响数据库正常使用的特殊事件。尽管数据库系统中采取了各种保护措施来防止数据库的安全性和完整性被破坏，保证并发事务的正确执行，但是计算机系统中硬件的故障、软件的错误、操作员的失误以及恶意的破坏仍是不可避免的。这些故障轻则造成运行事务非正常中断，影响数据库中数据的正确性；重则破坏数据库，使数据库部分或全部数据丢失。因此 DBMS 必须具有把数据库从错误状态恢复到某一正确状态的功能，这就是数据库的恢复。数据库管理系统中的恢复子系统是必不可少的，数据库系统所采用的恢复技术是否行之有效，不仅对系统的可靠程度起着决定性的作用，而且对系统的运行效率也有很大的影响，是影响系统性能的重要指标。

6.4.2 故障的种类

数据库系统中可能发生的故障大致可以分为以下几类。

1. 事务故障

事务故障是指事务没有达到预期的终点，使数据库可能处于不正确状态。事务内部更多的故障是非预期的，是不能由应用程序处理的，如运算溢出、并发事务发生死锁而被选中撤销该事务、违反了某些完整性限制等。恢复程序要在不影响其他事务运行的情况下，强行回滚事务，即撤销该事务已经做出的任何对数据库的修改，使得该事务好像根本没有启动一

样。这类恢复操作称为事务撤销。

2. 系统故障

系统故障是指造成系统停止运转的任何事故，使得系统要重新启动。例如，特定类型的硬件错误(CPU 故障)、操作系统故障、DBMS 代码错误、突然停电等。这类故障影响正在运行的所有事务，但不破坏数据库。这时主存内容、尤其是数据库缓冲区(在内存)中的内容都被丢失，所有运行事务都非正常终止。

一方面，发生系统故障时，一些尚未完成的事务的结果可能已存入物理数据库，从而造成数据库可能处于不正确的状态。为保证数据一致性，需要清除这些事务对数据库的所有修改，恢复子系统必须在系统重新启动时让所有非正常终止的事务回滚，强行撤销所有未完成的事务。另一方面，发生系统故障时，某些已完成的事务可能有一部分甚至全部留在缓冲区，尚未写回到磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失，这也会使数据库处于不一致状态，应将这些事务已提交的结果重新写入数据库。所以系统重新启动后，恢复子系统除了需要撤销所有未完成的事务外，还需要重做所有已提交的事务，以将数据库真正恢复到一致状态。

3. 介质故障

介质故障又称为硬故障，指外存故障，如磁盘损坏、磁头碰撞、瞬时强磁场干扰等。这类故障将破坏整个数据库或部分数据库，并影响正在存取这部分数据库的所有事务。

发生介质故障后，存储在磁盘上的数据被破坏，这时需要将发生介质故障之前的后援副本装入数据库，并重新做已成功完成的事务，将事务已提交的结果重新记入数据库。介质故障发生的可能性很小，但破坏性却是最大的，有时甚至会导致数据无法恢复。

4. 计算机病毒

计算机病毒是一种人为的故障或破坏，它像生物学所称的病毒一样可以繁殖和传播，并造成对计算机系统，包括数据库的破坏。

6.4.3 故障恢复

恢复就是利用存储在系统其他地方的备份数据来修复数据库中被破坏的或者不正确的数据。因此恢复机制涉及两个问题：一是建立备份数据；二是如何利用这些备份数据恢复数据库。建立备份数据最常用的技术是数据转储和登录日志文件。

1. 数据转储

所谓数据转储，就是周期性地把数据库复制到转储设备的过程。其中，转储设备是指用于放置数据库拷贝的磁带或磁盘。存放于转储设备中的备用的数据库文件称为后备副本。一旦系统发生介质故障，就可以使用后备副本来恢复数据库，但是使用后备副本恢复的数据库只能恢复到数据库转储时的状态。

(1) 静态转储和动态转储

转储可以分为静态转储和动态转储。

静态转储是在系统中无运行事务时进行的转储操作，即转储操作开始的时刻，数据库处于一致性状态，而转储期间不允许（或不存在）对数据库的任何存取、修改活动。显然，静态转储得到的一定是一个数据一致性的副本。

静态转储十分简单，但转储必须等待正在运行的用户事务结束后才能进行，而新的事务必须等待转储结束后才能执行。显然，这会降低数据库的可用性。

动态转储是指转储期间允许对数据库进行存取或修改，即转储和用户事务可以并发执行。

动态转储可以克服静态转储的缺点。它不用等待正在运行的用户事务结束，也不会影响新事务的运行。但是转储结束时后备副本上的数据并不能保证正确有效。为此，必须把转储期间各事务对数据库的修改活动登记下来，建立日志文件。这样，通过后备副本和日志文件就能把数据库恢复到某一时刻的正确状态。

（2）海量转储和增量转储

转储还可以分为海量转储和增量转储两种方式。

海量转储是指每次都转储全部数据库。使用海量转储得到的后备副本能够比较方便地进行数据库恢复。

增量转储是指每次只转储上一次转储后更新过的数据。增量转储适用于数据库较大，事务处理又十分频繁的数据库系统。

由于数据转储可以在静态和动态两种状态下进行，因此数据转储可以分为四类：动态海量转储、动态增量转储、静态海量转储和静态增量转储，如表 6-3 所示。

表 6-3 数据转储分类

		转储状态	
		动态转储	静态转储
转储方式	海量转储	动态海量转储	静态海量转储
	增量转储	动态增量转储	静态增量转储

利用转储得到的后备副本恢复数据库很方便，只需要将后备副本重新装入系统即可。但是，重装后备副本只能将数据库恢复到转储时的状态，要恢复到故障发生时的状态，则必须重新运行自转储以后的所有更新事务。图 6-5 所示为海量转储与恢复的过程系统在 T_1 时刻停止运行事务进行数据转储，在 T_2 时刻转储完毕，得到 T_2 时刻的数据库一致性的副本，系统运行到 T_3 时刻发生故障。为恢复数据库，首先重装后备副本，将数据库恢复到 T_2 时刻的状态，然后重新运行自 T_2 到 T_3 的所有更新事务，从而将数据库恢复到故障发生前的一致状态。

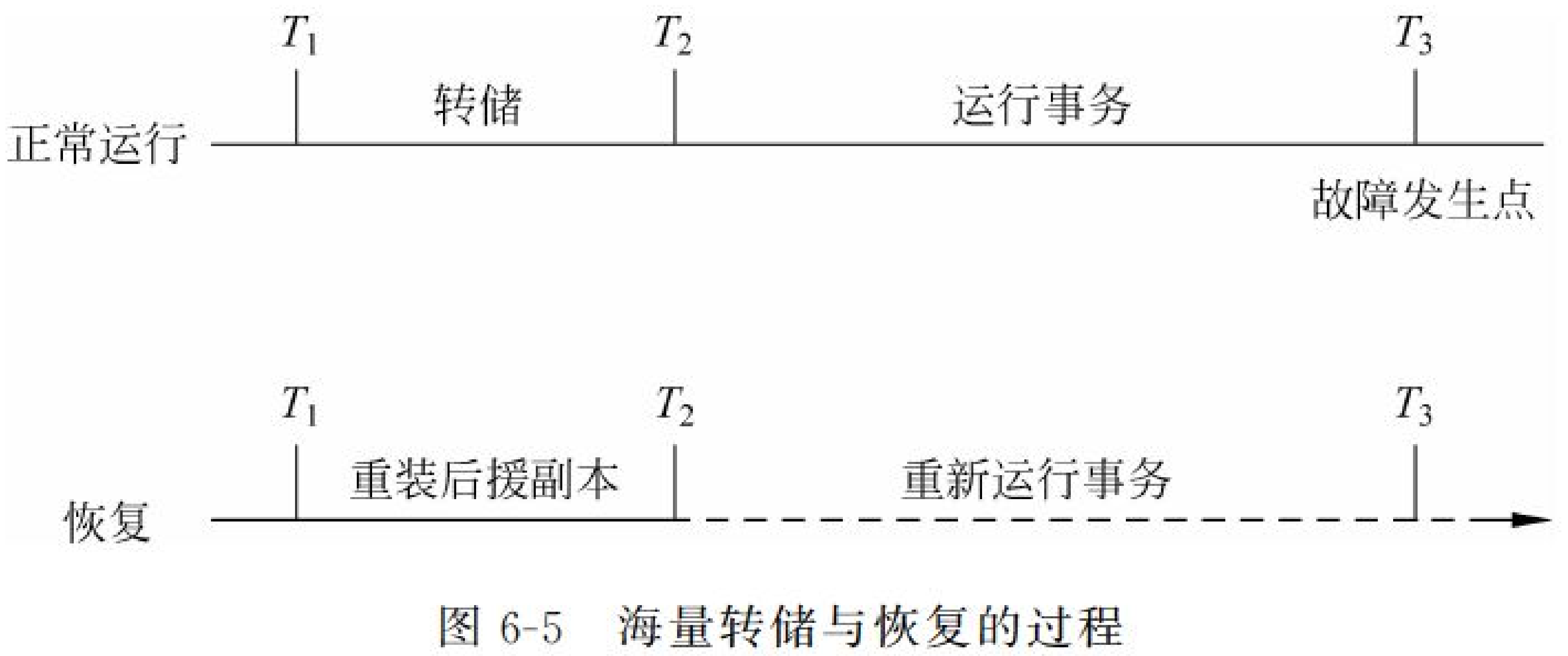

图 6-5 海量转储与恢复的过程

随着归档日志的堆积，恢复时间和对介质的占用都会随之增长。对于每一个企业，都有一个对增量恢复窗口可容忍的极限。因此，增量备份策略应该包含定期的数据库全备份，以便经常建立新的基点。

2. 日志文件

日志文件是用来记录每一次对数据库更新活动的文件。登记日志文件的目的是为数据库的恢复保留详细的数据。

(1) 日志文件的作用

日志文件可以用来进行事务故障恢复和系统故障恢复，并协助后备副本进行介质故障恢复。其具体作用如下：

- 事务故障恢复和系统故障恢复必须用日志文件。
- 在动态转储方式中，必须建立日志文件，后备副本和日志文件一起使用才能有效地恢复数据库。
- 在静态转储方式中，也可以建立日志文件。当数据库毁坏后可重新装入后备副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件，对已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤销处理。

(2) 登记日志文件

每次修改数据库时，都要登记日志文件。日志文件主要有两种格式：一种是以数据块为单位的日志文件，只要某个数据块中有数据被更新，就要将整个块更新前、更新后的内容存入日志文件；另一种是以记录为单位的日志文件，其中包括事务的开始和终止、事务的操作对象和操作类型、更新操作的数据旧值和新值等。

登记日志文件时必须遵循以下原则：

- 严格按并发事务执行的时间次序来进行登记。
- 必须先写日志文件，后写数据库。由于故障可能发生在对数据的修改写到数据库中和把表示这个修改的日志记录写到日志文件中的两个操作之间，如果先写了数据库修改，而在日志记录中没有登记这个修改，则以后就无法恢复这个修改了。如果先写了日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的撤销操作，并不会影响数据库的正确性。

(3) 恢复事务

发生故障时，首先根据前次转储的后备副本恢复数据库，然后利用日志文件进行恢复事务。

利用日志文件恢复事务主要分为两步：

① 从头扫描日志文件，找出哪些事务在故障发生时已经结束，哪些尚未结束。

② 对尚未结束的事务进行撤销(UNDO)处理，对已经结束的事务进行重做(REDO)处理。

撤销处理方法：反向扫描日志文件，对每个撤销的事务的更新操作执行反操作，即对已经插入的新记录执行删除，对已经删除的记录执行插入，恢复修改过的数据。

重做处理方法：正向扫描日志文件，对每个重做的事务，重新执行登记的事务的操作。

6.4.4 恢复策略

数据库系统的恢复包括事务故障恢复、系统故障恢复和介质故障恢复。对于不同的故障需要采用不同的恢复策略。

1. 事务故障恢复

事务故障是指事务在运行至正常终止点前被终止，这时恢复子系统应利用日志文件撤销此事务已对数据库进行的修改。其具体步骤如下：

① 反向扫描文件日志(即从最后向前扫描日志文件)，查找该事务的更新操作。

② 对该事务的更新操作执行逆操作。即将日志记录中"更新前的值"写入数据库。

- 插入操作，"更新前的值"为空，则相当于做删除操作。
- 删除操作，"更新后的值"为空，则相当于做插入操作。
- 若是修改操作，则相当于用修改前值代替修改后值。

③ 继续反向扫描日志文件，查找该事务的其他更新操作，并进行同样的处理。

④ 继续做下去，直到读到该事务的开始标记。

如果在该事务执行期间还有其他事务读了它的"废数据"，则对其他事务也要进行以上的处理，这可能会进一步引起其他事务的重新处理。

事务故障的恢复由系统自动完成，对用户是透明的，不需要用户干预。

2. 系统故障恢复

系统故障造成数据库不一致分为两种情况，一是未完成事务对数据库的更新可能已写入数据库；二是已提交事务对数据库的更新可能还留在缓冲区没有来得及写入数据库。因此恢复操作就是要撤销故障发生时未完成的事务，重做已完成的事务。

系统恢复的步骤如下：

① 正向扫描日志文件(即从头扫描日志文件)，找出故障发生前已经提交的事务(这些事务以 BEGIN TRANSACTION 开始，以 COMMIT 结束)，将其记入重做队列，并找出故障发生时尚未完成的事务(这些事务以 BEGIN TRANSACTION 开始，而没有以 COMMIT 结束)，将其记入撤销队列。

② 对撤销队列中的各个事务进行撤销。进行撤销的方法是，反向扫描日志文件，对每个撤销事务的更新操作执行逆操作，也就是将日志中的"更新前的值"写入数据库。

③ 对重做队列中的各个事务进行重做处理。进行重做处理的方法是，正向扫描日志文件，对每个重做事务重新执行日志文件登记的操作，将日志中的"更新后的值"写入数据库。

系统故障的恢复由系统自动完成，不需要用户干预。

3. 介质故障恢复

发生介质故障后，磁盘上的物理数据和日志文件都被破坏。恢复的方法是重装数据库，并重做已完成的事务。具体做法如下：

① 装入最新的数据库副本(离故障发生时刻最近的转储副本)，使数据恢复到最近一次

转储时的一致性状态，对于动态转储的数据库副本，还需要同时装入转储开始时刻的日志文件副本，利用系统故障恢复方法（即 REDO＋UNDO），将数据库恢复到一致性状态。对于静态转储的数据库副本，装入后数据库即处于一致性状态。对于动态转储的数据库副本，还要进行第②步。

② 装入相应的日志文件副本，重做已完成的事务。首先扫描故障发生时已提交的事务标识，将其记入重做队列，然后正向扫描日志文件，对重做队列中的所有事务进行重做处理，也就是将日志中的“更新后的值”写入数据库。

介质故障的恢复需要 DBA 介入，但是 DBA 只需要重装最近转储的数据库副本和有关的各日志文件副本，然后执行系统提供的恢复命令即可，具体的恢复操作仍由 DBMS 完成。

对于因为计算机病毒发生故障的数据库，要视具体情况而定，大多数都可以按照介质故障或系统故障恢复。

6.5 本章小结

DBMS 是管理数据的核心，其自身必须提供统一的数据保护功能以确保数据库的安全可靠和正确有效。本章介绍了数据库的安全性、完整性、并发控制和恢复四个方面的基本概念和实现方法。

数据库的安全性是指保护数据库，防止用户不合法使用所造成的数据泄露、修改或破坏。数据库的安全性可以通过用户标识与鉴别、存取控制（自主存取控制和强制存取控制）、视图机制、审计和数据加密等来实现。

数据库的完整性是指保证数据库数据的正确性、有效性和相容性，防止错误的数据进入数据库。完整性可以用完整性约束来表示。

数据库的并发控制以事务为单位，通常使用封锁技术实现并发控制。本章介绍了两类最常用的封锁：三级封锁协议和两段锁协议。三级封锁协议用于保证数据的一致性；可串行性是并行调度正确的唯一准则，两段锁协议是保证并行调度可串行性的封锁协议。对数据对象实施封锁，会带来活锁和死锁问题，并发控制机制必须提供适合数据库特点的解决方法。

数据库系统中可能发生的故障大致可以分为事务故障、系统故障、介质故障和计算机病毒几类。DBMS 必须具有把数据库从错误状态恢复到已知正确的某一状态的功能。数据转储和登记日志文件是数据库恢复中最常用的技术。

6.6 习题

6.6.1 名词解释

事务、数据库的安全性、封锁、共享锁、排他锁、活锁、死锁、封锁粒度、封锁协议、日志文件、事务故障、系统故障、介质故障、计算机病毒

6.6.2 简答题

1. 简述实现数据库安全性控制的常用方法和技术。
2. 数据库的完整性和安全性概念有什么区别和联系？
3. 数据库的完整性约束条件可以分为哪几类？
4. 试述事务的概念和四个属性。
5. 数据库的并发操作会带来哪些问题？如何解决？
6. 简述两种基本的封锁类型的含义。
7. 简述两段锁协议的含义。
8. 简述保证数据一致性的三级封锁协议。
9. 活锁是如何产生的？解决活锁的方法是什么？
10. 什么是死锁？避免死锁的方法有哪些？
11. 数据库运行中可能产生的故障有哪几类？
12. 数据库恢复的具体实现方法有哪些？
13. 简述数据库转储的含义和分类。
14. 日志文件的作用是什么？
15. 如何利用日志文件恢复事务？
16. 简述登记日志文件时必须遵循的原则。
17. 简述事务故障及其恢复策略。
18. 简述系统故障及其恢复策略。
19. 简述介质故障及其恢复策略。

6.6.3 综合题

试述某一个实际的 DBMS 产品中采用的恢复策略。

第7章 数据库设计

现代信息系统总是以一个数据库作为其核心和基础。因此，数据库的建立与使用水平代表着社会信息化建设的程度。设计一个数据库尤其是大型数据库是一项十分复杂而费时的工作，它不但要使用数据库和数据处理理论与技术，还涉及许多其他领域的知识，如管理科学、系统工程，当然还有计算机科学的许多相关领域。

7.1 数据库设计概述

数据库设计的目的就是实现数据库应用，所以它与数据库应用系统（即各种信息系统）的设计紧密相关，但二者又不完全相同。数据库设计者基于用户的各种应用需求（包括数据需求和处理需求）、选择适当的系统环境（硬件配置、操作系统和 DBMS 等）、使用合理的设计方法与技术来建立一个数据库以满足用户需要的过程称为数据库设计。为此，首先要明确数据库设计要考虑和解决的主要问题。

7.1.1 数据库设计的内容

数据库设计的内容主要包括数据库的结构特性设计、数据库的行为特性设计、数据库的物理模式设计。其中，数据库的结构特性设计起着关键作用，行为特性设计起着辅助作用。

1. 数据库的结构特性设计

数据库的结构特性设计是指数据库结构的设计，设计结果能否得到一个合理的数据模型，是数据库设计的关键。由于数据库的结构特性是静态的，一般情况下不会轻易变动，所以数据库的结构特性设计又称为数据库的静态结构设计。首先要将现实世界中的事物以及事物间的联系用 E-R 图表示出来，再对各个分 E-R 图进行汇总，得出数据库的概念结构模型，然后将概念结构模型转化为数据库的逻辑结构模型表示，最后进行数据库物理设计，并建立数据库。

2. 数据库的行为特性设计

数据库的行为特性设计是指应用程序、事务处理的设计。数据库的行为特性设计是基于数据库用户的行为和动作进行设计，而用户行为总是更新数据库内容的操作，用户行为特性是动态的，所以数据库的行为特性设计又称为数据库的动态特性设计。首先要将现实世

界中的数据用数据流程图和数据字典表示，并详细描述其中的数据操作要求，进而得出系统的功能模块结构和数据库的子模式。

3. 数据库的物理模式设计

数据库的物理模式设计要求是：根据库结构的动态特性(即数据库应用处理要求)，在选定的DBMS环境下，把数据库的逻辑结构模型加以物理实现，从而得出数据库的存储模式和存取方法。

在数据库设计中，通常将结构特性设计和行为特性设计结合起来，相互参照，同步进行，才能较好地达到设计目标。数据库设计者在进行设计时，应考虑到计算机的硬件环境和软件环境，考虑到当前以及未来时间段内对系统的需求，所设计的系统既能满足用户的近期需求，同时对远期的数据需求也具有相应的处理方案。也就是说，数据库设计者应充分考虑到系统可能的扩充和改动，使系统具有较长的生命周期。

7.1.2 数据库设计的方法

早期的数据库设计采用手工试凑法进行，使用手工试凑法设计数据库与设计人员的经验和水平有直接关系，它更像是一种技艺而不是工程技术。这种方法缺乏科学的理论和工程方法支持，数据库设计的质量很难得到保证，数据库常常是在投入使用以后才发现问题，因而不得不进行修改，这样就增加了系统维护的代价。随着计算机技术的飞速发展，人们也在不断努力探索各种非手工的数据库设计方法，并提出了多种数据库设计的准则和规范，这些设计方法被称为规范设计法。

非手工方法的主要代表有基于LRA方法、New Orleans方法、E-R模型方法等。其中New Orleans方法即新奥尔良法，是规范设计中比较著名的一种方法。它将数据库设计分为4个阶段：需求分析、概念设计、逻辑设计和物理设计。其后，许多科学家对此进行了改进，把数据库设计分为6个阶段进行，分别是需求分析、概念结构设计、逻辑结构设计、物理结构设计、数据库实施和数据库的运行与维护。本章节中所介绍的数据库设计的基本步骤正是按照新奥尔良法所提出的数据库设计的6个阶段进行操作的。

数据库工作者们十几年来一直致力于研究和开发数据库设计工具，目前已经具有一些比较成熟的数据库设计工具软件，从而减轻数据库设计人员的工作量，辅助他们更好地完成设计任务，特别是大型数据库的设计更需要自动设计工具的支持。

7.1.3 数据库设计的步骤

目前的数据库设计大多分6个阶段进行(如图7-1所示)。

1. 需求分析阶段

需求分析阶段是数据库设计的第一步，它是后继各阶段的基础，也是最困难、最耗时的一步。需求分析就是要准确了解并分析用户对系统的需求，并根据用户要求明确系统要达到的目标和要实现的功能。在分析设计阶段初期，用户可能会在原有基础上提出一些新的要求，设计者就要在需求分析阶段进行改善和完善，最大限度地满足用户需求。需求分析是

否准确和完善，直接影响整个数据库系统的性能。

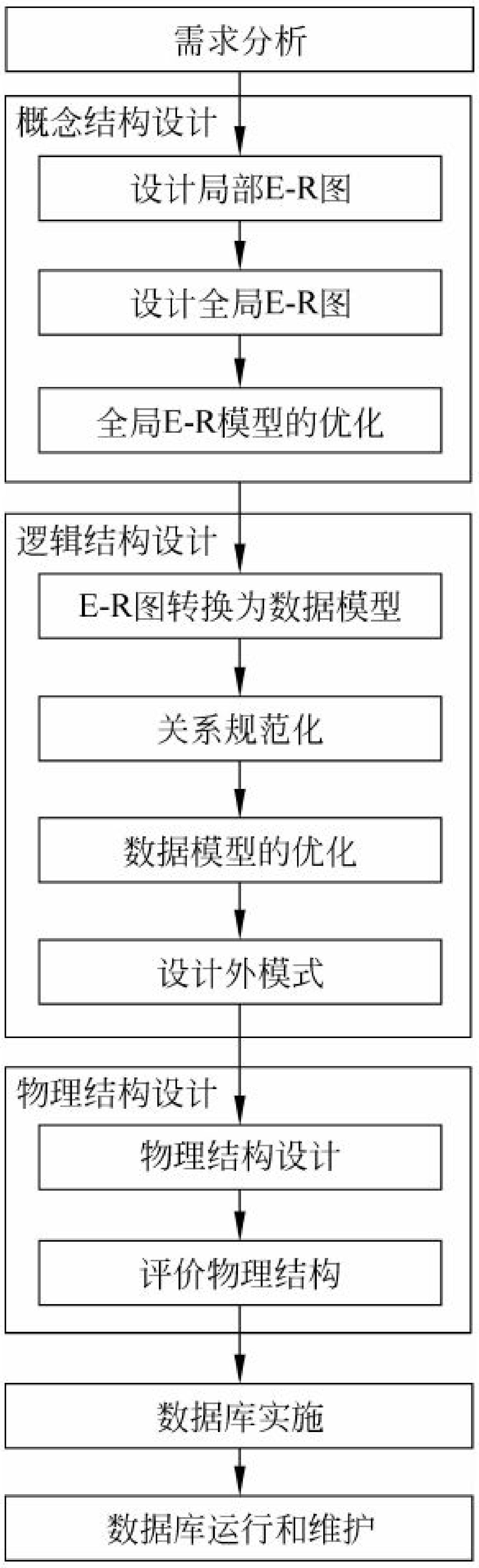

图 7-1　数据库设计的步骤

2. 概念结构设计阶段

这一阶段可以说是整个数据库设计的关键。设计者通过对用户需求进行归纳与抽象，形成一个与计算机硬件无关并独立于数据库管理系统的概念模型。

3. 逻辑结构设计阶段

逻辑结构设计阶段是将概念结构转换为具体数据库管理系统所支持的数据模型，并对数据模型进行优化。

4. 物理结构设计阶段

物理结构设计阶段是为逻辑设计模型选取一个适合应用环境的数据存储结构和存取方法，并评价设计，对系统性能进行预测。

5. 数据库实施阶段

在数据库实施阶段中，系统设计人员要根据数据库逻辑设计和物理设计结果编制与调试应用程序、装入数据，并进行数据库系统试运行。

6. 数据库运行和维护阶段

数据库应用系统在试运行结果满意的情况下，可投入正式运行。在系统运行过程中，必须不断地对其结构和性能进行评价、修正与完善，解决开发过程中遗留的问题，延长数据库系统的生命周期。

7.2 需求分析

需求分析就是调查、收集、分析、最后定义用户对数据库的各种要求。它是整个数据库设计的基础和出发点，其结果将直接影响后面各步的设计，甚至决定着最终设计的数据库的好坏与成败。为此，首先必须知道需求分析的任务是什么，以及采用什么样的方法进行需求分析。

7.2.1 需求分析的任务

需求分析的主要任务是详细调查现实世界的组织机构情况，充分了解系统概况和发展前景，明确用户的各种需求，收集支持系统目标的基础数据及其处理方法，确定新系统的功能和边界。

调查是需求分析的重要手段，只有通过对用户的调查研究，才能获取数据库系统所需要的数据情况和数据处理要求。调查的具体内容包括以下三个方面。

1. 数据库信息内容

信息需求定义了未来系统用到的所有信息，描述了数据之间本质上和概念上的联系，描述了实体、属性、组合及联系的性质。

2. 数据处理内容

数据处理需求中定义了用户要完成的数据处理的操作，描述了操作的优先次序、响应时间及数据处理的工作方式。

3. 数据安全性和完整性要求

要求定义数据的保密措施和存取控制要求，以及数据或数据间的约束限制。

7.2.2 需求分析的步骤和方法

实际上，确定用户的最终需求是一件非常困难的事情。因为一方面用户缺少计算机专业知识，不知道计算机能做什么不能做什么，因而不能准确地表达自己的需求；另一方面，设计人员缺少用户的专业知识，不易理解用户的真正需求，甚至可能会误解用户的需求。只有设计人员与用户加强交流，互相沟通，才能较好地完成需求分析。进行需求分析主要包括以下几个步骤。

1. 分析用户活动，产生用户活动图

这一步主要是了解现实社会的机构组织及用户当前的业务活动情况，搞清楚其业务流程，对一个比较复杂的处理，可划分为若干子处理，进行分析之后画出用户活动图。

2. 确定系统范围，产生系统范围图

这一步主要是确定系统的边界。明确哪些功能由人工完成，哪些功能由计算机实现。由计算机完成的功能就是新系统应该实现的功能。

3. 分析用户活动涉及的数据，产生数据流图

这一步主要是深入分析用户的业务处理，以数据流图的形式表示出数据的流向和对数据所进行的加工。

4. 分析系统数据，产生数据字典

数据流图仅仅表示出系统由哪几部分组成和各部分之间的关系，并没有说明各个成分的含义。只有对每个成分都给出确切定义后，才能完整地描述系统。

常用的调查方法有跟班作业、专家咨询、开调查会、请用户填写调查表、查阅相关数据记录等。

调查了解了用户的需求以后，需要进一步分析和表达用户的需求。分析和表达用户需

求的方法很多，常用的是结构化分析方法(Structure Analysis，SA 方法)，它是一种简单实用的方法。SA 方法从最上层的系统组织结构入手，采用自顶向下、逐层分解的方式分析系统。

7.2.3 需求分析注意的问题

需求分析是后面各个阶段设计的基础，关系到系统的设计是否合理和实用。这里要强调以下两点需注意的问题：

(1) 需求分析阶段一定要收集未来应用所涉及的数据。如果数据库设计人员仅仅按照当前应用来设计，新数据的加入就会在操作中显得十分困难，不仅会影响数据库的概念结构，而且将影响逻辑结构和物理结构。所以设计人员必须有前瞻性，充分考虑到未来应用可能发生的扩充和改变，使设计易于变动。

(2) 需求分析离不开用户的积极参与。由于用户缺少计算机专业知识，有时不能准确表达自己的要求；而设计人员缺少用户的专业知识，不易理解用户的真正需求，这就导致确定用户最终需求成为一件非常困难的事情。只有设计人员与用户加强沟通，互相交流，及时反馈用户意见，才能够较好地完成需求分析。因此，用户的积极参与是数据库设计中不可缺少的环节。

7.3 概念结构设计

在需求分析阶段，数据库设计人员充分调查并描述了用户的应用需求，但这些应用需求还是现实世界的具体需求，应该首先把它们抽象为信息世界的结构，这样才能更好地、更准确地用某一个 DBMS 实现用户的这些需求。将系统需求分析得到的用户需求抽象为信息结构的过程就是概念结构设计。它的目标是产生反映企业各组织信息需求的数据库概念结构，即概念模型。

概念结构独立于数据库逻辑结构，也独立于支持数据库的 DBMS。它是现实世界与机器世界的中介，它能够充分反映现实世界，包括实体与实体之间的联系，同时又易于向关系、网状、层次等各种数据模型转换。它是现实世界的一个真实模型，易于理解，便于和不熟悉计算机的用户交换意见，使用户易于参与，当现实世界需求改变时，概念结构又可以很容易地做相应的调整。因此概念结构设计是整个数据库设计的关键所在。

7.3.1 概念结构设计的方法与步骤

概念模型是数据模型的前身，它比数据模型更独立，更抽象，也更稳定。设计概念结构的策略主要有以下几种：

(1) 自顶向下：首先定义全局概念结构框架，然后逐步细化为完整的全局概念结构。

(2) 自底向上：首先定义每一局部应用的概念结构，然后按一定的规则把它们集成起来，从而得到全局概念结构。

(3) 逐步扩张：首先定义最重要的那些核心概念结构，再逐渐向外扩充生成其他概念结构，直至完成总体概念结构。

（4）混合策略：混合策略是把自顶向下和自底向上结合起来的方法，它先采用自顶向下的策略设计一个全局概念结构的框架，然后以它为骨架，再自底向上设计局部概念结构，并把它们集成起来。

其中最常用的策略是自底向上设计策略，其数据库概念设计的主要步骤如下：

① 进行数据抽象，设计局部概念模式。局部用户的信息需求是构建全局概念模式的基础。因此，首先要根据用户的需求为其建立相应的局部概念结构。

② 将局部概念模式综合成全局概念模式。在对局部概念模式进行综合的过程中，主要解决各局部模式对各种对象定义不一致的问题。把各个局部结构合并，还会产生冗余问题，或导致对信息需求的再调整与分析，以确定确切的含义。

③ 进行评审、改进。消除了所有冲突后，就可以把全局概念结构提交评审。评审分为用户评审和系统开发人员评审。前者的重点在于确认全局概念模式是否完整准确地反映了用户信息需求；而后者则侧重于确认全局结构是否完整，成分划分是否合理，是否存在不一致性等。如果在评审中发现问题，应及时改进。

7.3.2　数据抽象

进行数据抽象是概念设计的第一步。所谓抽象是对现实世界中的人、事、物的人为处理，抽取人们关心的共同特性，忽略非本质的细节，并把这些特性用各种概念精确地加以描述，形成某种模型。

抽象有两种形式，一种是系统状态的抽象，即抽象对象；另一种是系统转换的抽象，即抽象运算。在数据库设计中，概念设计的目的就是要定义抽象对象的关系结构。

数据抽象的三种基本方法是分类、聚集和概括。利用数据抽象方法可以对现实世界进行抽象，得出概念模型的实体集和属性。

1. 分类

所谓分类就是定义某一概念作为现实世界中一组成员的类型，这些对象具有某些共同的特性和行为，分类抽象了对象值和型之间的“成员”的语义。在 E-R 模型中实体集就是这种抽象。例如，在学校里，王老师是一名教师，他具有教师们共同的特性和行为：隶属于某教学系部，承担某门课程的教学任务。与王老师属同一对象的还有张老师、李老师等其他教师，如图 7-2 所示。

2. 聚集

聚集的数学意义就是笛卡儿积的概念。通过聚集，形成对象之间的一个联系对象，抽象了对象内部类型和对象内部组成部分的语义。属性的聚集组成了实体型。例如把实体集“教工”的“教工号”、“姓名”、“性别”等属性聚集在一起，形成实体型“教工”，如图 7-3 所示。

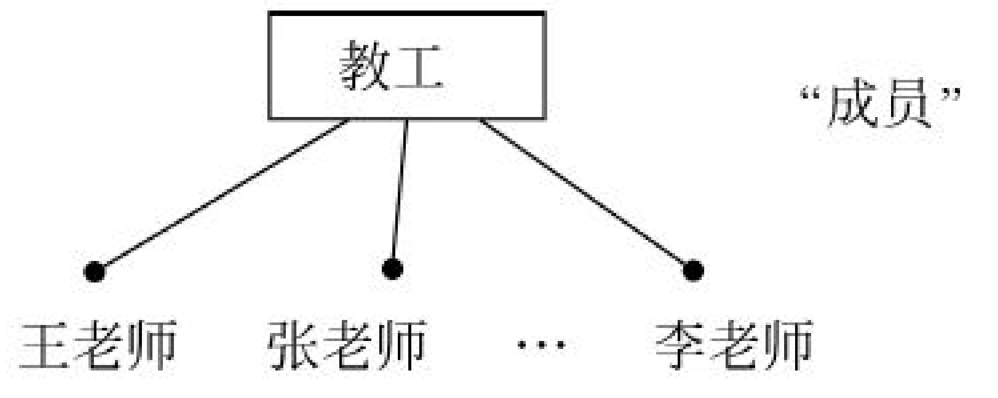

图 7-2　分类

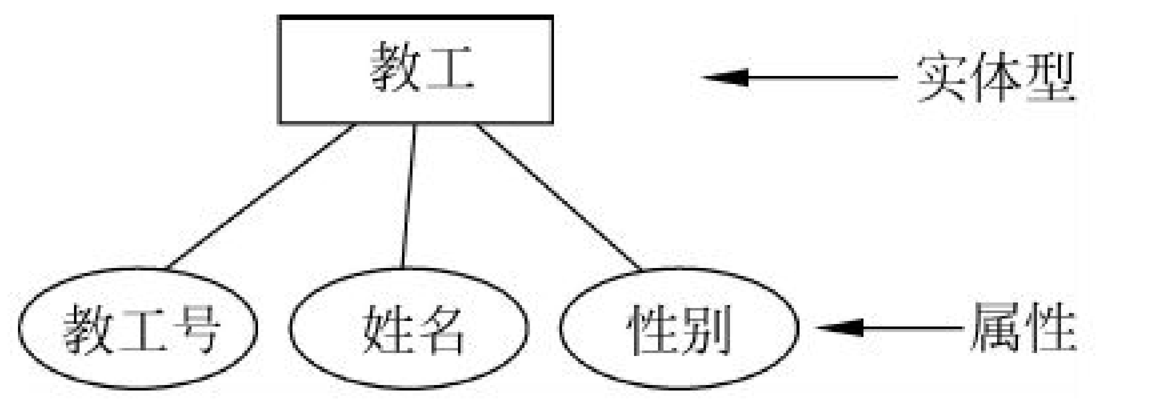

图 7-3　教工属性聚集实例

3. 概括

概括是从一类对象形成一个对象类型。例如一类对象{汽车,摩托车,自行车,…}可以概括为一个对象类型“公路车辆”,可以把公路车辆称为超类,把汽车等称为子类,如图 7-4 所示。概括的一个重要性质是继承性。继承性指子类继承超类的所有抽象。

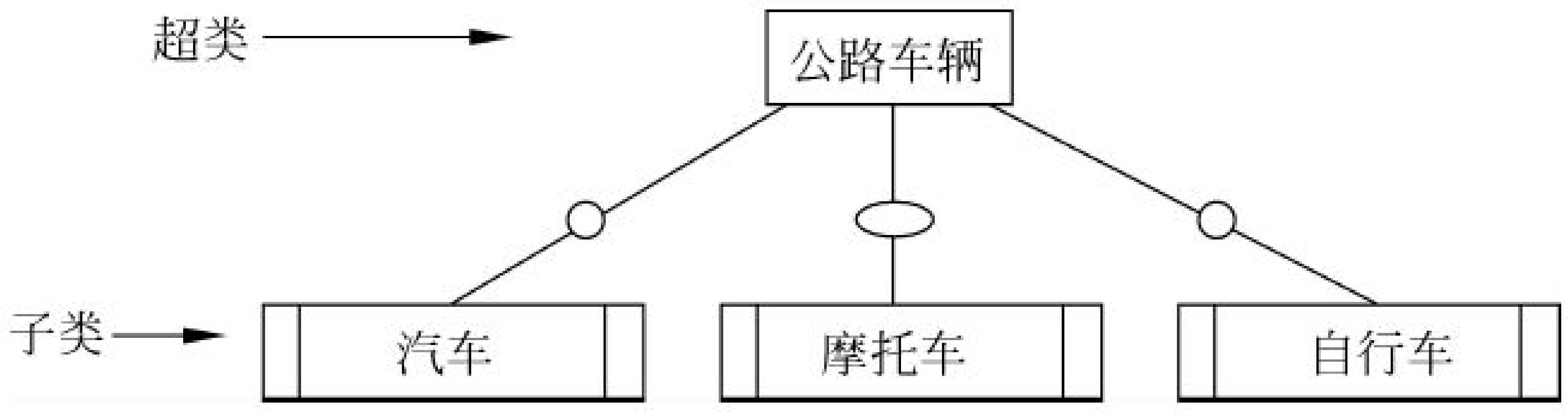

图 7-4 概括

7.3.3 采用 E-R 方法的数据库概念结构设计

概念设计方法最著名、最实用的是 P. P. S. Chen 于 1976 年提出的“实体-联系法”(Entity-Relationship Approach,简称 E-R 方法)。这种方法将现实世界的信息结构统一用属性、实体以及实体之间的联系,即 E-R 图来描述。

1. 设计局部 E-R 模型

通常,一个数据库系统都是为多个不同用户服务的。信息处理需求也会因为用户观点的不同而有所差异。在设计数据库概念结构时,一个有效的策略就是先分别考虑各个用户的信息需求,形成局部概念结构,然后再综合成全局结构。

局部 E-R 模型设计步骤如图 7-5 所示。

首先,确定局部结构的范围划分。划分的方式一般有两种:一种是依据系统的当前用户进行自然划分,另一种是按用户要求数据库提供的服务归纳成几类,使每一类应用访问的数据显著不同于其他类,然后为每类应用设计一个局部模型。局部结构范围的划分要自然且易于管理,范围之间的界面要清晰,彼此之间的相互影响要小,同时应注意范围的大小要适度。

接下来要标定实体、实体间的联系以及实体的属性。它们通常是按照对客观世界的理解和思维习惯,根据数据的逻辑关系来划分的。

划分实体和属性的基本准则要注意以下几点:

- 属性与它所描述的实体之间只能是单值联系,即联系只能是一对多的。
- 属性不能再有需要进一步描述的性质。
- 作为属性的数据项,除了它所描述的实体之外,不能再与其他实体具有联系。
- 能作为属性的数据应尽量作为属性处理。

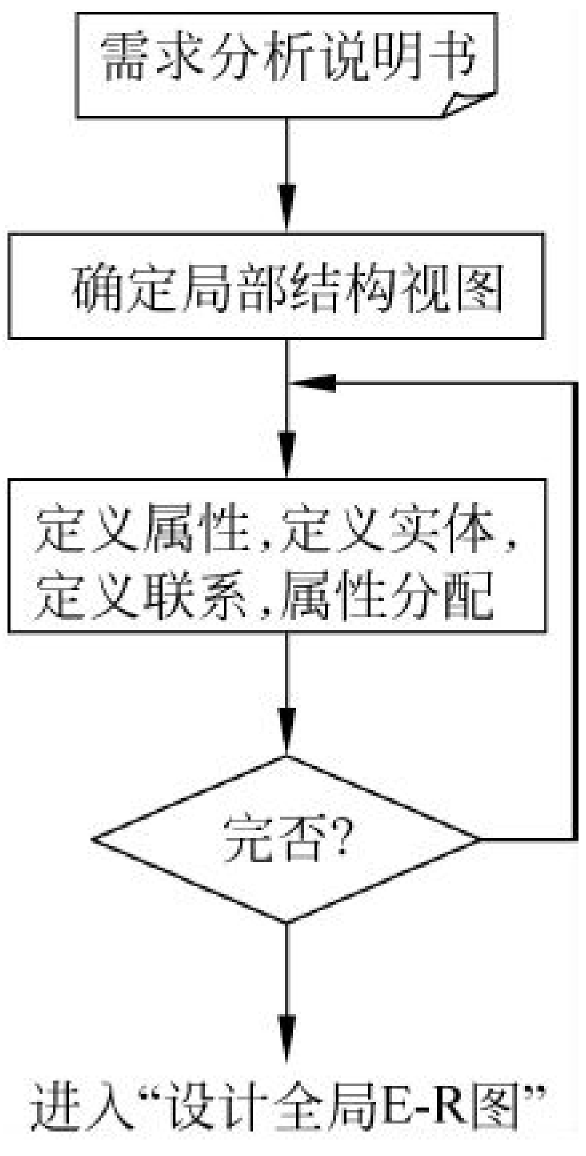

图 7-5 局部 E-R 图设计步骤

【例 7.1】 以仓库管理为例，描述设计 E-R 图的步骤。

步骤如下：

① 确定实体类型

本例中有项目 PROJECT、零件 PART 和零件供应商 SUPPLIER 三个实体类型。

② 确定联系类型

PROJECT 和 PART 之间是 $m:n$ 联系，即一个项目需要使用多种零件，一个零件可用于多个项目中。PART 和 SUPPLIER 之间也是 $m:n$ 联系，即一种零件可由多个供应商提供，一个供应商也可提供多种零件。分别定义联系类型为 P-P 和 P-S。

③ 确定实体类型的属性

实体类型 PROJECT 有属性：项目符号 J＃、项目名称 JNAME 、项目开工日期 DATE；实体类型 PART 有属性：零件编号 P＃、零件名称 PNAME、颜色 COLOR、重量 WEIGHT；实体类型 SUPPLIER 有属性：供应商编号 S＃、供应商名 SNAME、供应商地址 SADR。

④ 确定联系类型的属性

联系类型应该至少包括与之联系的所有实体类型的键，例如联系类型 P-P 有属性：需要的零件数量 TOTAL；联系类型 P-S 有属性：供应数量 QUANTITY。

⑤ 根据实体类型和联系类型画出 E-R 图（如图 7-6 所示）。

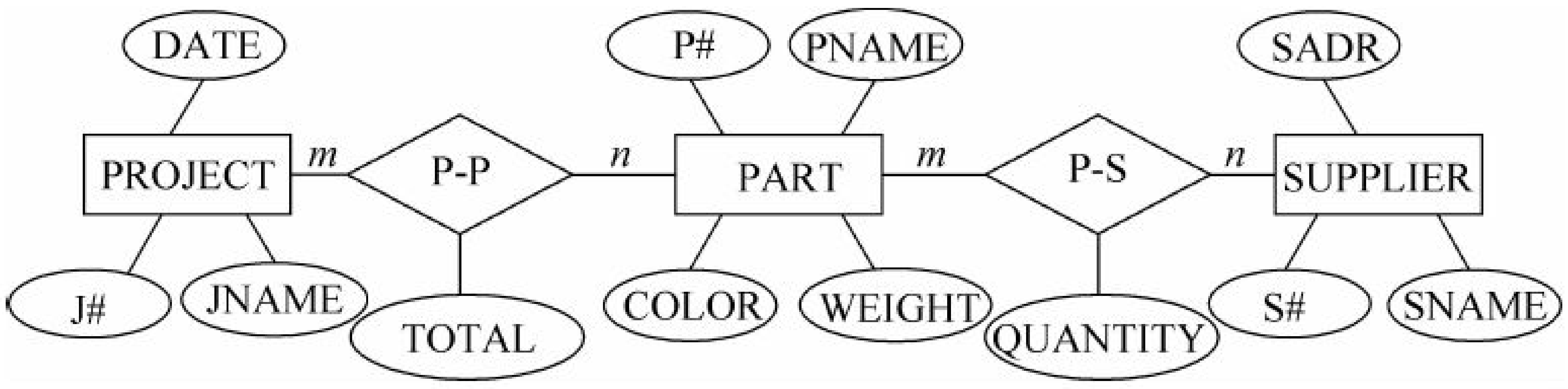

图 7-6 仓库 E-R 图

2. 设计全局 E-R 模型

这一步是将所有局部的 E-R 图集成为全局的 E-R 图，即全局的概念模型。设计全局概念模型的过程如图 7-7 所示。

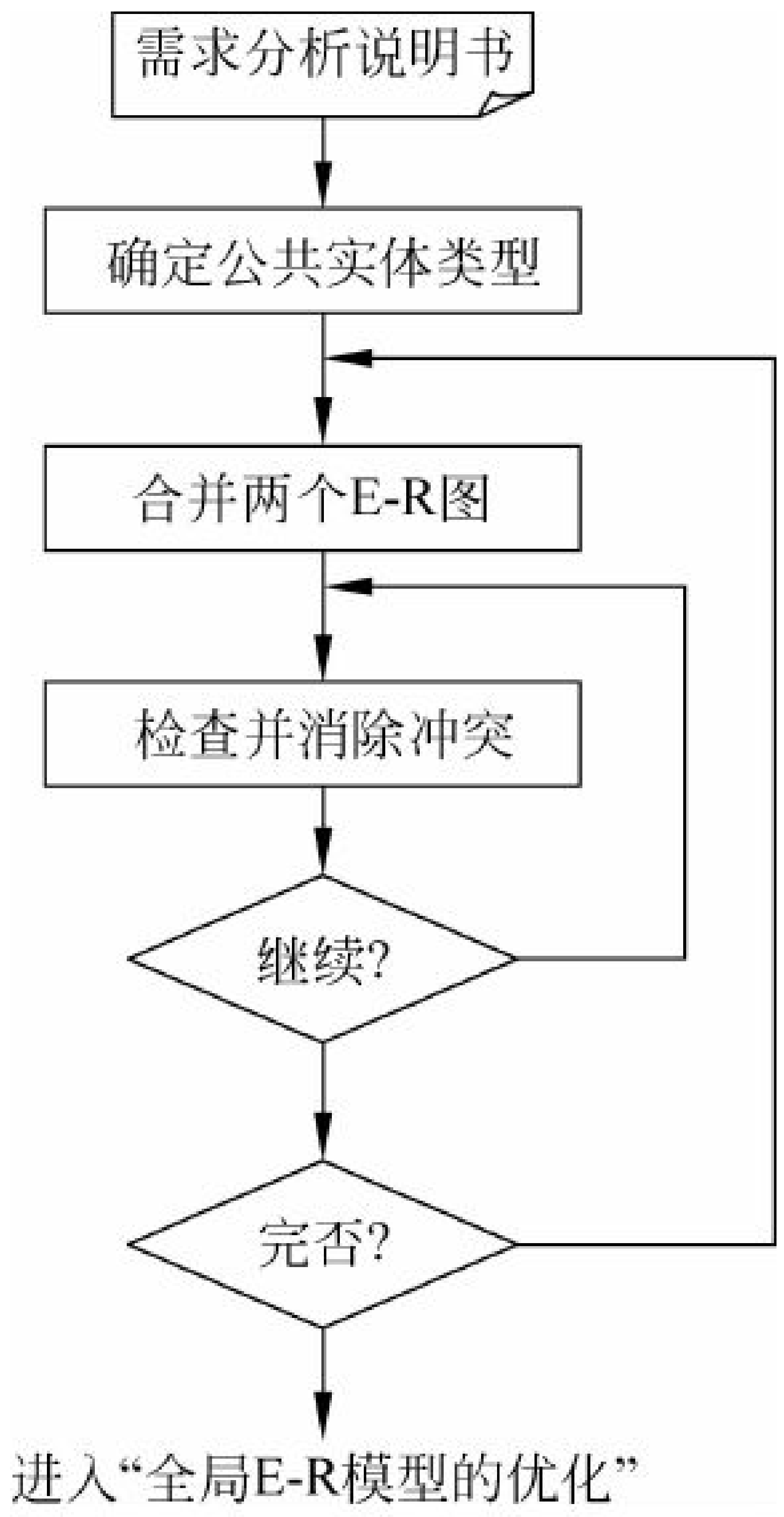

图 7-7 全局 E-R 图设计步骤

首先，要确定各局部结构中的公共实体类型。在这一步中，仅根据实体类型名和关键字来认定公共实体类型。一般把同名实体类型作为公共实体类型的一类候选，把具有相同键的实体类型作为公共实体类型的另一类候选。接下来就要把局部 E-R 图集成为全局E-R 图。

把局部 E-R 图集成为全局 E-R 图时，一般采用两两集成的方法，即先将具有相同实体的两个 E-R 图以该相同实体为基准进行集成，如果还有相同实体的 E-R 图，则再次集成，这样一直重复下去，直到所有具有相同实体的局部 E-R 图都被集成为止，从而得到全局的 E-R 图。

由于各类应用不同，不同的应用通常又由不同的人员设计成局部 E-R 模型，因此当将局部的 E-R 图集成为全局的

E-R 图时，不可避免地会有不一致的地方，称之为冲突。通常可能存在三类冲突，分别为属性冲突、命名冲突和结构冲突。

（1）属性冲突

① 属性域冲突，即属性值的类型、取值范围或取值集合不同。例如，由于学号是数字，因此某些部门（即局部应用）将学号定义为整数形式，而由于学号不用参与运算，因此另一些部门（即局部应用）将学号定义为字符型形式。又如，某些部门（即局部应用）以出生日期形式表示学生的年龄，而另一些部门（即局部应用）以整数形式表示学生的年龄。

② 属性取值单位冲突。例如，学生的身高，有的以米为单位，有的以厘米为单位，有的以尺为单位。

属性冲突通常用讨论、协商等行政手段加以解决。

（2）命名冲突

命名冲突通常包括同名异义和异名同义两种。

① 同名异义，即不同意义的对象在不同的局部应用中具有相同的名字。例如，局部应用 A 中将教室称为房间，局部应用 B 中将学生宿舍称为房间。

② 异名同义（一义多名），即同一意义的对象在不同的局部应用中具有不同的名字。例如，有的部门把教科书称为课本，有的部门把教科书称为教材。

命名冲突可能发生在实体、联系一级，也可能发生在属性一级。其中属性的命名冲突更为常见。处理命名冲突通常也像处理属性冲突一样，通过讨论、协商等行政手段加以解决。

（3）结构冲突

结构冲突有三种情况：

① 同一对象在不同应用中具有不同的抽象。例如，“课程”在某一局部应用中被当作实体，而在另一局部应用中被当作属性。

解决方法通常是把属性变换为实体或把实体变换为属性，使同一对象具有相同的抽象。

② 同一实体在不同局部视图中所包含的属性不完全相同，或者属性的排列次序不完全相同。

这是很常见的一类冲突，原因是不同的局部应用关心的是该实体的不同侧面。解决方法是使该实体的属性取各分 E-R 图中属性的并集，再适当设计属性的次序。例如，在局部应用 A 中，“学生”实体由学号、姓名、性别、平均成绩四个属性组成；在局部应用 B 中，“学生”实体由姓名、学号、出生日期、所在系、年级五个属性组成；在局部应用 C 中，“学生”实体由姓名、政治面貌两个属性组成；在合并后的 E-R 图中，“学生”实体的属性为：学号、姓名、性别、出生日期、政治面貌、所在系、年级、平均成绩。

③ 实体之间的联系在不同局部视图中呈现不同的类型。例如，实体 E1 与 E2 在局部应用 A 中是多对多联系，而在局部应用 B 中是一对多联系；又如在局部应用 X 中，E1 与 E2 发生联系，而在局部应用 Y 中，E1、E2、E3 三者之间有联系。

解决方法是根据应用的语义对实体联系的类型进行综合或调整。

3. 全局 E-R 图模型的优化

在得到全局 E-R 模型后，为了提高数据库系统的效率，还应进一步依据处理需求对全局 E-R 模型进行优化。

一个好的全局 E-R 图除了能反映用户功能需求外，还应满足下列条件：

- 实体类型个数尽可能少。
- 实体类型所含属性尽可能少。
- 实体类型间联系无冗余。

优化就是要达到这三个目的，但有时为了提高效率，根据具体情况可存在适当冗余。

7.4　逻辑结构设计

逻辑结构设计就是将概念结构转换为某个 DBMS 所支持的数据模型(如关系模型)，并对其进行优化。以关系模型为例，逻辑结构设计就是将概念结构(E-R 图)转换为一组关系模式，也就是将 E-R 图所表示的实体、实体的属性和它们之间的联系转化为关系模式，转换过程中可以采用下面的方法。

7.4.1　E-R 图转换为数据模型

逻辑结构设计将信息世界的概念模型 E-R 图转化为计算机世界的数据模型，需要遵循以下原则(以转换为关系模型为例)。

1. 实体转换为关系

每个实体都可以转换为一个单独的关系模式，实体的属性就是关系的属性，实体的码就是关系的码，如图 7-8 所示。

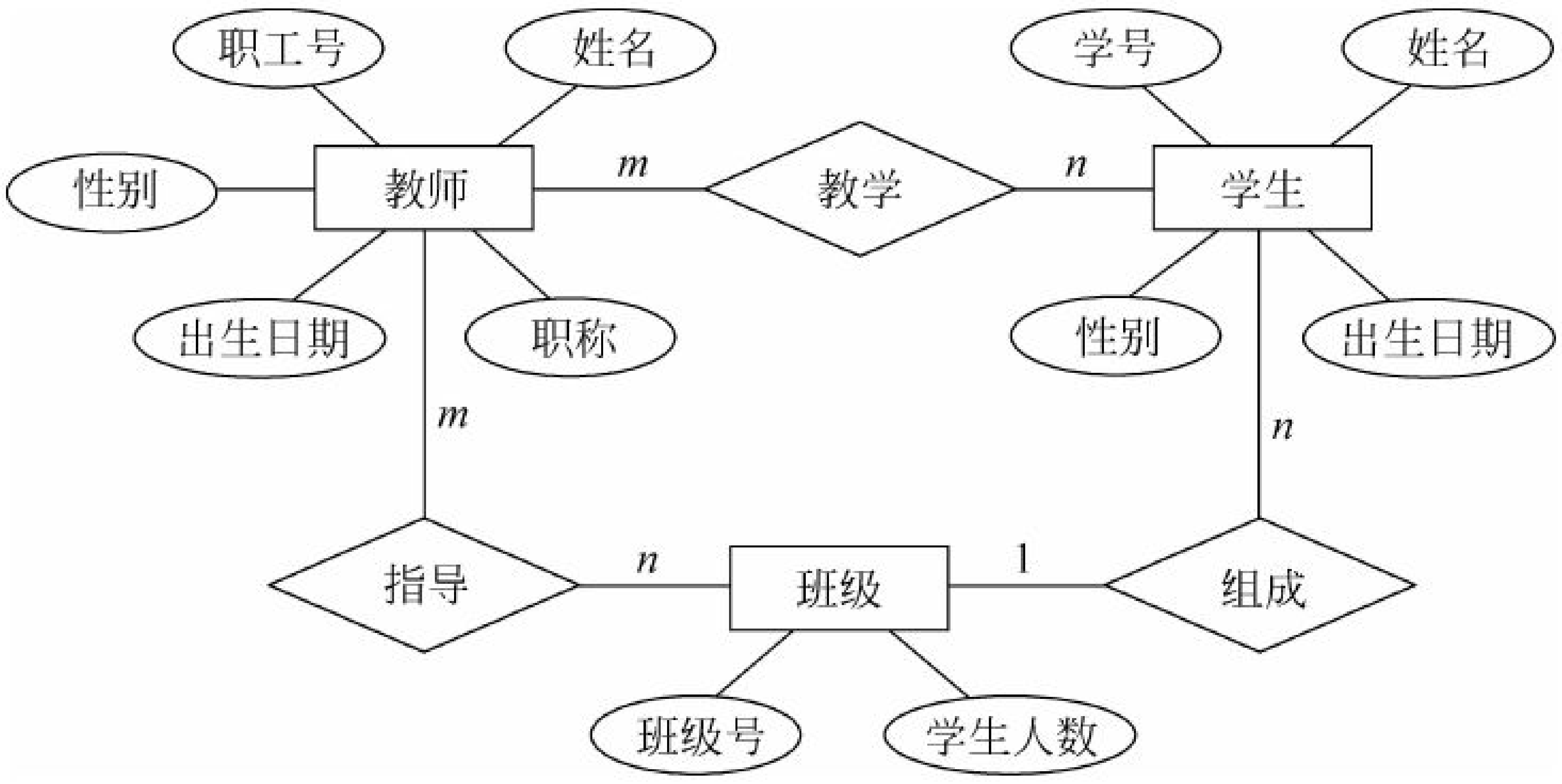

图 7-8　E-R 示例

E-R 图中的教师实体可以转换为下面的关系模式，其中“职工号”为关系的码(通常给属性加下划线表示码)，例如，教师(<u>职工号</u>，姓名，性别，出生日期，职称)。同样，学生和班级也可以转换为关系模式：学生(<u>学号</u>，姓名，性别，出生日期)和班级(<u>班级号</u>，学生人数)。

2. 联系转换为关系模式

(1) $m:n$ 型的联系

$m:n$ 型的联系可以转换为一个单独的关系模式，联系中各实体的码、联系本身的属性

一起转换为这个关系模式的属性，同时，将相关实体的码联合起来作为这个关系模式的码。依然以上面为例，“指导”联系是一个 $m:n$ 型的联系，可以将它转换为如下的关系模式，其中“职工号”和“班级号”联合作为关系的码：指导(职工号，班级号，考评成绩)。

（2）$1:n$ 型的联系

$1:n$ 型的联系有两种转换方法：

① 可以转换为一个单独的关系模式，联系中各实体的码、联系本身的属性一起转换为这个关系模式的属性，n 端实体的码作为关系的码。

② 可以合并到 n 端的关系模式中。

以上面的为例，“组成”联系为 $1:n$ 型的联系，一种方法是转换为一个单独的关系模式，以“学生”实体的码作为这个关系模式的码：组成(学号，班级号)。

另一种方法是将其与“学生”关系模式合并，这种方法更为合理，可以减少系统中关系的数量，提高查询效率，所以这种方法更可取。转换后，“学生”的关系模式为：学生(学号，姓名，性别，出生日期，班级号，学生人数)。

（3）$1:1$ 型的联系

$1:1$ 型的联系可以看做是 $1:n$ 型的联系一个特殊的情况，所以与 $1:n$ 型的转换类似。有两种方法：

① 可以转换为一个单独的关系模式，联系中各实体的码、联系本身的属性一起转换为这个关系模式的属性，同时，可以选择任一个实体的码为这个关系的候选码；

② 可以合并到任意一端的关系模式中，这时需要在该关系模式的属性中加入另一个关系模式的码和联系本身的属性。

上面的 E-R 图中没有这种类型的关系，可以这样构造，“学生”实体改成“班长”实体，一个班级只有一个班长，所以“班长”和“班级”之间的联系是 $1:1$ 型的联系。“班长”管理“班级”，所以它们之间存在“管理”的联系。按照上面的转换方法，可以转换成一个单独的关系模式，以其中任意一个实体的码为候选码。关系模式为：管理(班长学号，班级号)或 管理(班级号，班长学号)。在实际应用中，更倾向于选择第二种方法。可以与“班长”实体的关系模式合并，并且在关系模式中加入“班级”实体的码，表示为：班长(班长学号，姓名，性别，出生日期，班级号)。同样，如果与“班级”关系模式合并，则为：班级(班级号，学生人数，班长学号)。

（4）多元联系转换为关系模式

将三个或三个以上的多元联系转换为一个单独的关系模式。与这个多元联系相关的各实体的码以及联系的属性转换为关系的属性，相关各实体的码组合为关系的码。例如，“住宿”联系是一个三元联系，可以将它转换为如下关系模式，其中“学生号”、“宿舍号”和“床位号”联合作为关系的组合码：住宿(学生号，宿舍号，床位号)。

（5）实体自联系转换为关系模式

同一实体集上的联系称为自联系，可以参照实体之间的联系进行转换。

3．具有相同码的关系模式可合并

如果两个关系模式具有相同的主码，可以考虑将它们合并为一个关系模式。合并方法是将其中一个关系模式的全部属性加入到另一个关系模式中，然后去掉其中的同义属性(可

能同名也可能不同名)，并适当调整属性的次序。

假如有一个学生关系：学生(学号，姓名，性别，出生日期)；还有一个所属班级关系：所属班级(学号，班级)。由于它们的主码都为“学号”，所以可以将它们合并到一个关系模式中：学生(学号，姓名，性别，出生日期，班级)。

按照上述方法，图7-8所示的E-R图可以转换为下面的关系模式：

学生(学号，姓名，性别，出生日期，班级号)

班级(班级号，学生人数)

教师(职工号，姓名，性别，出生日期，职称)

教学(职工号，学号)

指导(职工号，班级号)

7.4.2 关系规范化

关系规范化是指将E-R图转换为数据模型后，通常以规范化理论为指导，对关系进行分解或合并，这是关系模式的初步优化。一般按照以下步骤进行：

① 考察关系模式的函数依赖关系。按照需求分析得到的语义关系，将各个关系模式中的函数依赖关系提炼出来，并进行极小化处理，消除冗余。

② 按照数据依赖理论，将关系模式分解，至少达到第三范式，即消除部分函数依赖和传递依赖。并不是规范化程度越高关系就越优，因为规范化程度越高，系统就会越经常做连接运算，这是以牺牲效率为代价的。一般来说，达到第三范式就足够了。

7.4.3 数据模型的优化

为了进一步提高数据库应用系统的性能，还可以对产生的关系模式进行进一步优化，即修改、调整和重构。根据需要，可以添加适当的“冗余”，以提高效率。例如，实体班级和学生的关系为$1:n$，经过规范化后，学生中加入属性“所在班级”，通过对相同班号的学生求和就可以得到某班的人数，可是这种求班级人数的工作是经常做的，而且对于班级来说，其人数也是非常重要的属性，因此可以在班级中加入属性“人数”，虽然这是冗余字段，可是能提高数据库的整体效率，这是很值得的。如果一个关系的某个属性可以由本关系的其他属性计算得到，那么这绝对是冗余属性；如果一个关系的某个属性是由其他关系的属性计算得来的，则这个属性在某些时候可以加在此关系中。

7.4.4 设计外模式

前面几个阶段设计出的关系模式是系统的模式，为了实现数据和应用程序的独立性，在逻辑结构设计阶段还要根据数据库系统的模式设计出外模式(也称子模式或用户模式)。外模式是保护数据库安全性的一个有力措施。每个用户只能看见和访问所对应的外模式中的数据，数据库中的其余数据对他们来说是不可见的。同时，对于每一个外模式，数据库系统都有一个外模式/模式映像，它定义了该外模式与模式之间的对应关系。这些映像定义通常包含在各自外模式的描述中。当模式改变时(如增加新的数据类型、新的数据项、新的关系等)，由数据库管理员对数据库外模式/模式映像做相应的改变，可以使外模式保持不变。从

而应用程序不必修改,保证了数据的逻辑独立性。

在设计外模式时,要注意以下几点:

- 按照用户习惯进行命名,包括关系名、属性名。外模式与模式的属性本质即使相同也可以取不同的名字。
- 针对用户的不同级别定义不同的外模式,以保护系统的安全性。
- 构造必要的外模式,以简化用户操作。

7.5 数据库物理设计

数据库的物理设计是对于给定的逻辑数据模型选取一个最适合应用环境的物理结构的过程。数据库的物理结构是指数据库在物理设备上的存储结构与存取方法,它依赖于给定的计算机系统。此外物理设计还包括物理数据库结构对运用需求的满足,如存储空间、存取策略方面的要求、响应时间及系统性能方面的要求等。

数据库的物理设计可以分两步进行:

① 确定数据的物理结构,即确定数据库的存取方法和存储结构。

② 对物理结构进行评价。

对物理结构进行评价的重点是时间和效率。如果评价结构满足原设计要求,则可以进行物理实施;否则应该重新设计或修改物理结构,有时甚至要返回逻辑设计阶段修改数据模型。

7.5.1 数据库物理设计的内容和方法

由于不同的数据库产品所提供的物理环境、存取方法和存储结构各不相同,供设计人员使用的设计变量、参数范围也各不相同,所以数据库的物理设计没有通用的设计方法可遵循,仅有一般的设计内容和设计原则供数据库设计人员参考。

数据库设计人员都希望自己设计的物理数据库结构能满足事务在数据库上运行时响应时间短、存储空间利用率高和事务吞吐率大的要求。为此,设计人员应该对要运行的事务进行详细的分析,获得选择物理数据库设计所需要的参数,并且应当全面了解给定的 DBMS 的功能、DBMS 提供的物理环境和工具,尤其是存储结构和存取方法。

数据库设计者在确定数据存取方法时,必须清楚三种相关的信息:

(1) 数据库查询事务的信息,它包括查询所需要的关系、查询条件所涉及的属性、连接条件所涉及的属性、查询的投影属性等信息。

(2) 数据库更新事务的信息,它包括更新操作所需要的关系、每个关系上的更新操作所涉及的属性、修改操作要改变的属性值等信息。

(3) 每个事务在各关系上运行的频率和性能要求。

例如,某个事务必须在 5s 内结束,这对于存取方法的选择有直接影响。这些事务信息会不断地发生变化,所以数据库的物理结构要能够做适当的调整,以满足事务变化的需要。

关系数据库物理设计的内容主要指选择存取方法和存储结构,包括确定关系、索引、聚簇、日志、备份等的存储安排和存储结构,确定系统配置等。

7.5.2 关系模式存取方法的选择

由于数据库是为多用户共享的系统，它需要提供多条存取路径才能满足多用户共享数据的需求。数据库物理设计的任务之一就是确定建立哪些存取路径和选择哪些数据存取方法。关系数据库常用的存取方法有索引方法、聚簇方法和 HASH 方法等。

1. 索引存取方法的选择

选择索引存取方法实际上就是根据应用要求确定对关系的哪些属性列建立索引，对哪些属性列建立组合索引，对哪些建立唯一索引等。选择索引方法的基本原则是：

(1) 如果一个属性经常在查询条件中出现，则考虑在这个属性上建立索引；如果一组属性经常在查询条件中出现，则考虑在这组属性上建立组合索引。

(2) 如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引。

(3) 如果一个属性经常在连接操作的连接条件中出现，则考虑在这个属性上建立索引；同理，如果一组属性经常在连接操作的连接条件中出现，则考虑在这组属性上建立索引。

(4) 关系上定义的索引数要适当，并不是越多越好，因为系统要为维护索引付出代价，也要为查找索引付出代价。例如，更新频率很高的关系上定义的索引，数量就不能太多，因为更新一个关系时，必须对这个关系上有关的索引做相应的修改。

2. 聚簇存取方法的选择

为了提高某个属性或属性组的查询速度，把这个属性或属性组上具有相同值的元组集中存放在连续的物理块上的处理称为聚簇，这个属性或属性组称为聚簇码。

(1) 建立聚簇的必要性

聚簇功能可以大大提高按聚簇码进行查询的效率。例如要查询计算机系所有学生的名单，假设计算机系有 200 名学生，在极端情况下，这 200 名学生所对应的数据元组分布在 200 个不同的物理块上。尽管对学生关系已按所在系建立了索引，由索引会很快找到计算机系学生的元组标识，避免了全表扫描。然而再由元组标识去访问数据块时就要存取 200 个物理块，执行 200 次 I/O 操作。如果将同一系的学生元组集中存放，则每读一个物理块就可以得到多个满足查询条件的元组，从而可以显著地减少访问磁盘的次数。聚簇功能不但适用于单个关系，而且适用于经常进行连接操作的多个关系，即把多个连接关系的元组按连接属性值聚集存放，聚集中的连接属性称为聚簇码。这就相当于把多个关系按“预连接”的形式存放，从而大大提高了连接操作的效率。

(2) 建立聚簇的基本原则

一个数据库可以建立多个聚簇，但一个关系只能加入一个聚簇。选择聚簇存取方法就是确定需要建立多少个聚簇，确定每个聚簇中包括哪些关系。聚簇设计时可以分两步进行：先根据规则确定候选聚簇，再从候选聚簇中去除不必要的关系。

设计候选聚簇的原则是：

① 对经常在一起进行连接操作的关系可以建立聚簇。

② 如果一个关系的一组属性经常出现在相等、比较条件中，则该单个关系可建立聚簇。

③ 如果一个关系的一个(或一组)属性上的值重复率很高,则此单个关系可建立聚簇。也就是说对应每个聚簇码值的平均元组不能太少,如果太少了,聚簇的效果不明显。

④ 如果关系的主要应用是通过聚簇码进行访问或连接,而其他属性访问关系的操作很少,则可以使用聚簇。尤其是当 SQL 语句中包含有与聚簇有关的 ORDER BY、GROUP BY、UNION、DISTINCT 等字句或短语时,使用聚簇特别有利,可以省去对结果集的排序操作。反之,当关系较少利用聚簇码操作时,最好不要使用聚簇。

检查候选聚簇,取消其中不必要关系的方法是:

① 从聚簇中删除经常进行全表扫描的关系。

② 从聚簇中删除更新操作远多于连接操作的关系。

③ 不同的聚簇中可能包含相同的关系,一个关系可以在某一个聚簇中,但不能同时加入多个聚簇。要从这多个聚簇方案(包括不建立聚簇)中选择一个较优的,其标准是在这个聚簇上运行各种事务的总代价最小。

(3) 建立聚簇应注意的问题

建立聚簇时,应注意三个问题:

① 聚簇虽然提高了某些应用的性能,但是建立与维护聚簇的开销是相当大的。

② 对已有的关系建立聚簇,将导致关系中的元组移动其物理存储位置,这样会使关系上原有的索引无效,要想使用原索引就必须重建原有索引。

③ 当一个元组的聚簇码值改变时,该元组的存储位置也要做相应地移动,所以聚簇码值应当相对稳定,以减少修改聚簇码值所引起的维护开销。

7.5.3 确定数据库的存储结构

确定数据的存储位置和存储结构要综合考虑存取时间、存储空间利用率和维护代价三方面的因素。这三个方面常常相互矛盾,需要进行权衡,选择一个折中的方案。

1. 确定数据的存放位置

为了提高系统性能,应该根据应用情况将数据的易变部分与稳定部分、经常存取部分和存取频率较低部分分开存放。对于有多个磁盘的计算机,可以采用下面几种存取位置的分配方案。

(1) 将表和索引放在不同的磁盘上,这样在查询时,由于两个磁盘驱动器并行工作,可以提高物理 I/O 读写的效率。

(2) 将比较大的表分别存放在两个磁盘上,以加快存取速度,这在多用户环境下特别有效。

(3) 将日志文件、备份文件与数据库对象(表、索引等)放在不同的磁盘上,以改进系统的性能。

(4) 对于经常存取或对存取时间要求高的对象(如表、索引)应放在高速存储器(如硬盘)上,对于存取效率小或对存取时间要求低的对象(如数据库的数据备份和日志文件备份等只在故障恢复时才使用),如果数据量很大,可以存放在低速存储设备上。

由于各个系统所能提供的对数据进行物理安排的手段、方法差异很大,因此设计人员应该仔细了解给定的 DBMS 提供的方法和参数,针对具体应用环境的要求,对数据进行适当的物理安排。

2. 确定系统配置

DBMS产品一般都提供了一些系统配置变量和存储分配参数供设计人员和DBA对数据库进行物理优化。在初始情况下，系统都为这些变量赋予了合理的默认值。但是这些默认值不一定适合每一种应用环境。在进行数据库的物理设计时，还需要重新对这些变量赋值，以改善系统的性能。

系统配置变量很多，例如同时使用数据库的用户数、同时打开的数据库对象数、内存分配参数、缓冲区分配参数（使用的缓冲区长度、个数）、存储分配参数、物理块的大小、物理块装填因子、时间片大小、数据库的大小和锁的数目等，这些参数值影响存取时间和存储空间的分配。进行物理设计时需要根据应用环境确定这些参数值，以使系统性能最佳。

进行物理设计时对系统配置变量的调整只是初步的，在系统运行时还要根据实际运行情况做进一步的参数调整，以提升系统性能。

3. 评价物理结构

在设计过程中，效率问题的考虑只能在各种约束得到满足且确定方案可行之后进行。下面介绍一下物理设计的性能。

多性能测量方面设计者能灵活地对初始设计过程和未来的修整做出决策。假设数据库性能用“开销”(Cost)，即时间、空间及可能的费用来衡量，则在数据库应用系统生存期中，总的开销包括规划开销、设计开销、实施和测试开销、操作开销和运行维护开销。

对物理设计者来说，主要考虑操作开销，即为使用户获得及时、准确的数据所需的开销和计算机资源的开销。可分为如下几类：

(1) 查询和响应时间：响应时间定义为从查询开始到查询结果开始显示之间所经历的时间，它包括CPU服务时间、CPU队列等待时间、I/O队列等待时间、封锁延迟时间和通信延迟时间。

一个好的应用程序设计可以减少CPU服务时间和I/O服务时间。例如，有效地使用数据压缩技术，选择好访问路径和合理安排记录的存储等，都可以缩短服务时间。

(2) 更新事务的开销：主要包括修改索引、重写物理块或文件、校验等方面的开销。

(3) 报告生成的开销：主要包括检索、重组、排序和结果显示方面的开销。

(4) 主存储空间开销：包括程序和数据所占用的空间的开销。一般对数据库设计者来说，可以对缓冲区分配(包括缓冲区个数和大小)做适当的调整，以减少空间开销。

(5) 辅助存储空间：分为数据块和索引块两种空间。设计者可以控制索引块的大小、装载因子、指针选择项和数据冗余度等。

实际上，数据块设计者能有效控制I/O服务和辅助空间；有限地控制封锁延迟，CPU时间和主存空间；而完全不能控制CPU和I/O队列等待时间，以及数据通信延迟时间。

7.6 数据库实施

对数据库的物理设计初步评价完成后就可以开始建立数据库了。数据库实施主要包括以下几个方面的内容：

- 用 DDL 定义数据库结构；
- 组织数据入库；
- 编制与调试应用程序；
- 数据库试运行。

7.6.1 定义数据结构

确定了数据库的逻辑结构与物理结构后，就可以用所选用的 DBMS 提供的数据定义语言(DDL)来严格描述数据库结构。

例如，可以用 SQL 语句定义如下表结构：

```
CREATE TABLE 学生
 (学号 StudentNo(8),
  …
);
CREATE TABLE 课程
 (课程号 CourseNo(8),
  …
);
接下来是在这些基本上定义视图:
CREATE VIEW …
 (
  …
)
…
```

7.6.2 数据装载

数据库结构建立好后，就可以向数据库中装载数据了。组织数据入库是数据库实施阶段最主要的工作。

对于数据量不是很大的小型系统，可以用人工方法完成数据的入库，其步骤如下：

① 筛选数据。需要装入数据库中的数据通常都分散在各个部门的数据文件或原始凭证中，所以首先必须把需要入库的数据筛选出来。

② 转换数据格式。筛选出来的需要入库的数据，其格式往往不符合数据库要求，还需要进行转换，这种转换有时可能很复杂。

③ 输入数据。将转换好的数据输入计算机中。

④ 校验数据。检查输入的数据是否有误。

对于大中型系统，由于数据量极大，用人工方式组织数据入库将会耗费大量人力物力，而且很难保证数据的正确性。因此应该设计一个数据输入子系统，由计算机辅助数据的入库工作。其步骤如下：

① 筛选数据。

② 输入数据。由录入人员将原始数据直接输入到计算机中。数据输入子系统应提供输入界面。

③ 校验数据。数据输入子系统采用多种检验技术检查输入数据的正确性。

④ 转换数据。数据输入子系统根据数据库系统的要求，从录入的数据中抽取有用成分，对其进行分类，然后转换数据格式。抽取、分类和转换数据是数据输入子系统的主要工作，也是数据输入子系统的复杂性所在。

⑤ 综合数据。数据输入子系统根据系统的要求将转换好的数据进一步综合成最终数据。要完成转换数据、综合数据两项工作，直接将老系统中的数据转换成新系统中需要的数据格式即可。

为了保证数据能够及时入库，应在进行数据库物理设计的同时编制数据输入子系统。

7.6.3 编制与调试应用程序

数据库应用程序的设计应该与数据库设计并行进行。在数据库实施阶段，当数据库结构建立好后，就可以开始编制与调试数据库的应用程序了，也就是说，编制与调试应用程序是与组织数据入库同步进行的。调试应用程序时由于数据入库尚未完成，可先使用模拟数据。

7.6.4 数据库试运行

应用程序调试完毕，并且已有一小部分数据入库后，就可以开始数据库的试运行。数据库试运行也称联合调试，其主要工作包括以下几个方面。

1. 功能测试

即实际运行应用程序，执行对数据库的各种操作，测试应用程序的各种功能。

2. 性能测试

即测量系统的性能指标，分析是否符合设计目标。

数据库物理设计阶段在评价数据库结构估算时间、空间指标时，做了许多简化和假设，忽略了许多次要因素，因此结果必然很粗糙。数据库试运行则是要实际测量系统的各种性能指标（不仅是时间、空间指标），如果结果不符合设计目标，则需要返回物理设计阶段，调整物理结构，修改参数；有时甚至需要返回逻辑设计阶段，调整逻辑结构。

重新设计物理结构、调整逻辑结构，会导致数据重新入库。由于数据入库的工作量实在太大，因此可以采用分期输入数据的方法，即先输入一小批量数据供先期联合调试使用，待试运行基本合格后再输入大批量数据，逐步增加数据量，逐步完成运行评价。

在数据库试运行阶段，由于系统不稳定，硬、软件故障随时都有可能发生。而系统的操作人员对新系统还不熟悉，误操作也不可避免，因此必须做好数据库的转储和恢复工作，尽量减少对数据库的破坏。

7.6.5 数据库其他设计

其他设计工作包括对数据库的安全性、完整性、一致性和可恢复性等的设计。这些设计总是以牺牲效率为代价的，设计人员的任务就是要在效率和尽可能多的功能之间进行合理权衡。

1. 数据库的再组织

设计者对数据库的概念、逻辑和物理结构的改变称为再组织。再组织通常是由于环境需求的变化或性能原因引起的。一般，数据库管理系统都提供数据库的再组织应用程序。

2. 故障恢复方案设计

数据库设计中考虑的故障恢复方案，一般都是基于数据库管理系统提供的故障恢复手段的。如果数据库管理系统已提供了完善的软硬件故障恢复和存储介质的故障恢复手段，那么设计阶段的任务就简化为确定系统登录的物理参数、缓冲区个数、大小、逻辑块的长度、物理设备等。否则，就要指定人工备份方案。

3. 安全性考虑

许多数据库管理系统都有描述各种对象(如记录、数据项)的存取权限的成分，在设计时，根据对用户需求的分析，规定相应的存取权限。子模式是实现安全性要求的一个重要手段。也可以在应用程序中设置密码，对不同的使用者分配一定的密码，以密码控制使用级别。

4. 事务控制

大多数数据库管理系统都支持事务概念，以保证多用户环境下的数据完整性和一致性。事务控制有人工和系统两种控制办法，系统控制以数据操作语句为单位，人工控制则由程序员以事务的开始和结束语句显示实现。大多数 DBMS 提供封锁粒度的选择，封锁粒度一般有表级、页面级、记录级和数据项级，粒度越大控制越简单，但并发控制性能差，这些在设计中都要运筹考虑。

7.7 数据库运行和维护

数据库试运行结果符合设计目标后，数据库就可以真正投入运行了。数据库投入运行标志着开发任务的基本完成和维护工作的开始，并不意味着设计过程的终结。由于应用环境在不断变化，数据库运行过程中物理存储也会不断变化，对数据设计进行评价、调整、修改等维护工作是一个长期的任务，也是设计工作的继续和提高。

在数据库运行阶段，对数据库经常性的维护工作主要是由 DBA 完成的，它包括以下内容。

7.7.1 数据库的转储与恢复

数据库的转储和恢复是系统正式运行后最重要的维护工作之一，DBA 要针对不同的应用要求指定不同的转储计划，定期对数据库和日志文件进行备份，以保证一旦发生故障，能利用数据库备份及日志文件备份，尽快将数据库恢复到某种一致性状态，并尽可能减少对数据库的破坏。

7.7.2 数据库的安全性与完整性维护

DBA 必须对数据库的安全性和完整性控制负起责任。根据用户的实际需要授予不同的操作权限。此外，在数据库运行过程中，由于应用环境的变化，对安全性的要求也会发生变化，比如有的数据原来是机密，现在是可以公开查询了，而新加入的数据又可能是机密了。而系统中用户的密级也会变化，这些都需要 DBA 根据实际情况修改原有的安全性控制。同样，由于应用环境的变化，数据库的完整性约束条件也会发生变化，也需要 DBA 不断修正，以满足用户要求。

7.7.3 数据库性能的监督与改进

在数据库运行过程中，监督系统运行，对监测数据进行分析，找出改进系统性能的方法是 DBA 的又一重要任务。目前，许多 DBMS 产品都提供了检测系统性能参数的功能，DBA 可以利用这些工具方便地得到系统运行过程中一系列性能参数的值。DBA 应该仔细分析这些数据，判断当前系统是否处于最佳运行状态，如果不是，则需要通过调整某些参数来进一步改进数据库性能。

7.7.4 数据库的功能完善

数据库运行一段时间后，由于记录不断被增、删、改，会使数据库的物理存储变坏，从而降低数据库存储空间的利用率和数据的存取效率，使数据库的性能下降。这时 DBA 就要对数据库进行重组织，或部分重组织（只对频繁增、删的表进行重组织）。数据库的重组织不会改变原设计的数据逻辑结构和物理结构，只是按原设计要求重新安排存储位置，回收垃圾，减少指针链，提高系统性能。DBMS 一般都提供了重组织数据库使用的应用程序，帮助 DBA 重新组织数据库。

当数据库应用环境发生变化时，例如，增加新的应用或新的实体，取消某些已有应用，改变某些已有应用，这些都会导致实体及实体间的联系也发生相应的变化，使原有的数据库设计不能很好地满足新的需求，从而不得不适当调整数据库的模式和内模式。例如，增加新的数据项，改变数据项的类型，改变数据库的容量，增加或删除索引，修改完整性约束条件等。这就是数据库重构造。DBMS 都提供了修改数据结构的功能。

重构造数据库的程度是有限的，如果应用变化太大，已无法通过重构数据库来满足新的需求，或重构数据库的代价太大，则表明现有数据库应用系统的生命周期已经结束，应该重新设计新的数据库系统，开始新数据库应用系统的生命周期了。

7.8 本章小结

本章主要讨论数据库设计的方法和步骤，详细介绍了数据库设计中规划、需求分析、概念设计、物理设计及运行与维护各个阶段的目标、方法和应注意的事项。其中，概念结构设计和逻辑结构设计是整个数据库设计过程中最重要的两个环节，也是本章重点介绍的内容。

在本章的学习中，不但要通过书中介绍的理论和实例掌握数据库设计的基本方法，还要

学习在实际工作中运用这些思想，设计出符合应用需求的数据库应用系统。

7.9 习题

7.9.1 名词解释

需求分析、概念结构设计、逻辑结构设计、物理结构设计、分类、聚集

7.9.2 简答题

1. 简述数据库设计过程。
2. 简述数据库设计过程各个阶段的设计描述。
3. 试述数据库设计方法。
4. 数据库设计的需求分析阶段主要任务是什么？调查的内容是什么？
5. 需求分析阶段应注意哪些问题？
6. 简述概念设计的设计策略和具体步骤。
7. 什么是数据抽象？主要有哪两种形式的抽象？
8. 试述采用 E-R 方法的数据库概念设计的过程。
9. 简述逻辑结构设计阶段的主要内容。
10. 简述数据库物理设计的内容和步骤。
11. 数据库运行和维护阶段主要有哪些工作？

7.9.3 综合题

某医院病房计算机管理系统需要如下信息：

科室：科名，科电话，科地址，医生姓名；

病房：病房号，床位号，所属科室名；

医生：姓名，职称，年龄，所属科室名，工作证号；

病人：病历号，姓名，性别，诊断，主管医生，病房号。

其中，一个科室有多个病房、多个医生，一个病房只能属于一个科室，一个医生只属于一个科室，但可负责多个病人的诊治，一个病人的主管医生只有一个。

(1) 用 E-R 图表示该系统的概念模型。

(2) 将得到的 E-R 图转化为等价的关系模型，并指出每个关系模式的主码。

第8章 数据库编程

8.1 嵌入式 SQL

当数据库设计好并建立后，就可以着手开发前台的应用程序了。本章将详细讲述如何用编程工具或语言来访问、连接以及操纵后台数据库。

8.1.1 嵌入式 SQL 的特点

SQL 是一种双重式语言，它既是一种用于查询和更新的交互式数据库语言，又是一种应用程序进行数据库访问时所采用的编程式数据库语言，即 SQL 具有交互式与嵌入式两种形式：

- 交互式 SQL：一般 DBMS 都提供联机交互工具，用户可直接输入 SQL 命令对数据库进行操作，由 DBMS 来进行解释。
- 嵌入式 SQL：能将 SQL 语句嵌入到高级语言中，使应用程序充分利用 SQL 访问数据库的能力和高级语言的过程处理能力，一般需要预编译，将嵌入的 SQL 语句转化为高级语言编译器能处理的语句。嵌入 SQL 的高级语言称为主语言或宿主语言。

SQL 语言在这两种方式中的大部分语法是相同的。嵌入式 SQL 在编写访问数据库的程序时，必须从普通的编程语言开始(如 C 语言)，再把 SQL 加入到程序中。所以，嵌入式 SQL 语言就是将 SQL 语句直接嵌入到程序的源代码中，与其他程序设计语言语句混合。专用的 SQL 预编译程序将嵌入的 SQL 语句转换为能被程序设计语言(如 C 语言)的编译器识别的函数调用。然后，C 编译器编译源代码为可执行程序。各个数据库厂商都采用嵌入 SQL 语言，并且都符合 ANSI/ISO 的标准。所以，如果采用合适的嵌入 SQL 语言，那么可以使得程序能够在各个数据库平台上执行，同时，每个数据库厂商又扩展了 ANSI/ISO 的标准，提供了一些附加的功能。这样，也使得每个数据库产品在嵌入 SQL 方面有一些区别。

8.1.2 SQL 语言和宿主语言编程

1. 嵌入 SQL 与宿主语言编程的特点

以 SQL Server 数据库和 C 语言作为宿主语言为例，嵌入 SQL 编程的基本特点如下：

(1) 程序的开头 EXEC SQL INCLUDE SQLCA;引入用于应用程序和数据库之间的通

信的数据结构 SQLCA。在 SQLCA 中的 SQLCODE 返回 SQL 语句执行后的结果状态。根据 SQLCODE 判断下一步的流程。SQLCODE 会在后续的内容中介绍。

(2) 在 BEGIN DECLARE SECTION 和 END DECLARE SECTION 之间定义了宿主变量。宿主变量可被 SQL 语句引用,也可以被 C 语言语句引用。它用于将程序中的数据通过 SQL 语句传给数据库管理器,或从数据库管理器接收查询的结果。

(3) 每次访问数据库之前必须进行 CONNECT 操作,连接到某一个数据库上。这时,应该保证数据库服务及实例已经启动。

(4) 执行语句。每条嵌入式 SQL 语句都由 EXEC SQL 开始,表明它是一条 SQL 语句。每一条嵌入 SQL 语句都有结束符号,例如在 C 语言中是“;”。这也是告诉预编译器在 EXEC SQL 和“;”之间是嵌入 SQL 语句。

(5) 断开数据库的连接。

下面是一个简单的嵌入式 SQL 语言的程序,在 VC 下需要设置项目导入 sqlserver 安装目录下的 sqlakw32.lib 和 caw32.lib 类库。下面的例子将演示如何在嵌入式 SQL 中创建表,插入数据,并显示数据。student 表结构如表 8-1 所示。

表 8-1

字 段 名	类 型	约 束
SNO	char(5)	主键
SNAME	char(10)	
AGE	int	
CITY	char(10)	

【例 8.1】 编写嵌入式 SQL 语言程序以完成创建 student 表、插入四条测试数据,并显示学生的全部信息。

```
#include <stdio.h>
#include <stdlib.h>
EXEC SQL INCLUDE sqlca;
int main()
{
   EXEC SQL BEGIN DECLARE SECTION;
   //主变量
   char sno[5];
   char sname[10];
   int age;
   char city[10];
   EXEC SQL END DECLARE SECTION;
   printf("这是一个嵌入式程序的测试\n");
   EXEC SQL CONNECT TO mysqlserver.school USER sa.abc;
   //连接到数据库,其中 computername 是机器名,school 是数据库名 ,
   //用户名是 sa,密码是 abc
   if (SQLCODE = = 0)
   {
      printf("连接数据库 ");
   }
```

```
    else
    {
        // 连接数据库错误
        printf("连接数据库错误 ");
        return (1);
    }
        //执行创建表
        EXEC SQL create table student (
            sno  char(5)  primary key,
            sname  char(10)  not null,
            age  int,
            city  char(10)
        );
        //执行插入数据
        EXEC SQL insert into student values('S1','张三',20,'天津');
        EXEC SQL insert into student values('S2','李四',10,'北京');
        EXEC SQL insert into student values('S3','王五',15,'天津');
        EXEC SQL insert into student values('S4','赵六',20,'天津');
        //错误处理
        EXEC SQL WHENEVER SQLERROR GOTO error;
        EXEC SQL WHENEVER NOT FOUND GOTO done;
        //定义游标
        EXEC SQL DECLARE myclass CURSOR FOR
        SELECT sname, city FROM student;

        EXEC SQL OPEN myclass;

        for ( ; ; ) {
        /* 在结果集中取出下一行 */
        EXEC SQL FETCH myclass INTO :sname, :city;
        /* 显示数据 */
        printf ("sname: %s ",sname);
        printf ("城市: %s\n",city);
        }
        error: //错误处理代码块
        //打印出错误信息
        printf ("SQL error %d\n", sqlcode);
        done:
        /* 关闭游标 */
        EXEC SQL WHENEVER SQLERROR continue;
        EXEC SQL CLOSE myclass;
        //断开连接
        EXEC SQL DISCONNECT ALL;
        return 0;
}
```

2. 常用的嵌入 SQL 语句

表 8-2 所示为常用的嵌入式 SQL 语句。

表 8-2 常用的嵌入式 SQL 语句

关 键 字	关 键 字
BEGIN DECLARE SECTION	PREPARE
CLOSE	SELECT INTO
CONNECT TO	SET ANSI_DEFAULTS
DECLARE CURSOR	SET CONCURRENCY
DELETE (POSITIONED)	SET CONNECTION
DELETE (SEARCHED)	SET CURSOR_CLOSE_ON_COMMIT
DESCRIBE	SET CURSORTYPE
DISCONNECT	SET FETCHBUFFER
END DECLARE SECTION	SET OPTION
EXECUTE	SET SCROLLOPTION
EXECUTE IMMEDIATE	UPDATE (POSITIONED)
FETCH	UPDATE (SEARCHED)
GET CONNECTION	WHENEVER
OPEN	

部分语句将在本章后面做介绍。嵌入式 SQL 语句分为静态 SQL 语句和动态 SQL 语句两类。静态 SQL 语句在编译时已经生成执行计划。而动态 SQL 语句只有在执行时才产生执行计划。动态 SQL 语句首先执行 PREPARE 语句要求数据库管理系统分析、确认和优化语句，并为其生成执行计划。

8.1.3 静态 SQL 编程

1. 声明变量

主变量就是在嵌入式 SQL 语句中引用主语言中说明的程序变量。声明嵌入 SQL 语句中使用的 C 变量，它的声明方法如下：

```
EXEC SQL BEGIN DECLARE SECTION;
   char first_name[50];
   char last_name[] = "White";
EXEC SQL END DECLARE SECTION;
…
EXEC SQL SELECT au_fname INTO :first_name
   from authors where au_lname = :last_name;
…
```

在嵌入式 SQL 语句中使用主变量前，必须采用 BEGIN DECLARE SECTION 和 END DECLARE SECTION 对主变量进行说明。这两条语句不是可执行语句，而是预编译程序的说明。主变量是标准的 C 程序变量。嵌入 SQL 语句使用主变量来输入数据和输出数据。C 程序和嵌入 SQL 语句都可以访问主变量。

在以 SQL 为基础的 DBMS 支持的数据类型与程序设计语言支持的数据类型之间有很大差别。这些差别对主变量影响很大。一方面，主变量是一个用程序设计语言的数据类型说明并用程序设计语言处理的程序变量；另一方面，在嵌入 SQL 语句中用主变量保存数据

库数据。所以，在嵌入 SQL 语句中，必须映射 C 数据类型为合适的 SQL Server 数据类型。必须慎重选择主变量的数据类型。在 SQL SERVER 中，大多数数据类型都能够自动转换。具体内容请参看 SQL Server 帮助文档。

2. 连接数据库

在程序中，使用 CONNECT TO 语句来连接数据库。该语句的完整语法为：

```
CONNECT TO {[server_name.]database_name} [AS connection_name] USER [login[.password] |
$integrated]
```

其中：

- server_name 为服务器名。如省略，则为本地服务器名。
- database_name 为数据库名。
- connection_name 为连接名。可省略。如果仅仅使用一个连接，那么无须指定连接名。可以使用 SET CONNECTION 来使用不同的连接。
- login 为登录名。
- password 为密码。

在"EXEC SQL CONNECT TO mysqlserver.pubs USER sa.abc;"中，服务器是 mysqlserver，数据库为 pubs，登录名为 sa，密码为 abc。

3. 数据操作

可以使用 SELECT INTO 语句查询数据，并将数据存放在主变量中。请看下面的例子：

```
EXEC SQL SELECT SNAME INTO :student_name
   FROM student WHERE CITY = :student_city;
```

使用 DELETE 语句删除数据。其语法类似于 Transact-SQL 中的 DELETE 语法。如：

```
EXEC SQL DELETE FROM student WHERE sname = '张三'
```

使用 UPDATE 语句可以更新数据。其语法就是 Transact-SQL 中的 UPDATE 语法。如：

```
EXEC SQL UPDATE student SET age = age + 1 WHERE sname = '李四'
```

使用 INSERT 语句可以插入新数据。其语法就是 Transact-SQL 中的 INSERT 语法。如：

```
EXEC SQL INSERT INTO student VALUES('S4','赵六',20,'天津');
```

以上介绍的是单行数据的调用方法，多行数据的查询和修改需要用到本节第 5 部分介绍的游标。

4. SQLCA

应用程序执行时，每执行一条 SQL 语句，就返回一个状态符和一些附加信息。这些信息反映了 SQL 语句的执行情况，它有助于用户分析应用程序的错误所在。这些信息都存放

在 sqlca.h 的 sqlca 结构中。如果一个源文件中包含 SQL 语句，则必须要在源程序中定义一个 SQLCA 结构，而且名为 SQLCA。预编译器自动在嵌入 SQL 语句中包含 SQLCA 数据结构。在程序中可以使用 EXEC SQL INCLUDE SQLCA，目的是告诉 SQL 预编译程序在该程序中包含一个 SQL 通信区。也可以不写，系统会自动加上 SQLCA 结构。

SQLCODE 是结构体 SQLCA 的一个数据分量。DBMS 是 SQL 的通讯区即 SQLCA 向应用程序报告运行错误信息。SQLCA 是一个含有错误变量和状态指示符的数据结构。通过检查 SQLCA，应用程序能够检查出嵌入式 SQL 语句是否成功，并根据成功与否决定是否继续往下执行。

SQLODE 是结构体 SQLCA 的一个数据分量，也是 SQLCA 结构中最重要的部分在执行每条嵌入式 SQL 语句时，DBMS 在 SQLCA 中设置变量 SQLCODE 值，以指明语句的完成状态：

- =0，该语句成功执行，无任何错误或报警。
- <0，出现了严重错误。
- >0，出现了报警信息。

编程过程中，要随时通过判断 SQL 语句的执行返回值来决定下一步的流程。

5. 游标的使用

用嵌入式 SQL 语句查询数据分为两类情况：一类是单行结果，一类是多行结果。对于单行结果，可以使用 SELECT INTO 语句；对于多行结果，则必须使用 cursor 即游标来完成。游标是一个与 SELECT 语句相关联的符号名，它使用户可逐行访问由 SQL Server 返回的结果集。使用游标时，需要按照以下步骤进行：

① 声明游标：

如：
```
EXEC SQL DECLARE C1 CURSOR FOR
        SELECT id, name, dept, job, years, salary, comm FROM staff;
```

② 打开游标

如：
```
EXEC SQL OPEN c1;
```

完整语法为：

```
OPEN 游标名 [USING 主变量名 | DESCRIPTOR 描述名]
```

关于动态 OPEN 游标的描述见 8.1.4 节动态 SQL 编程。

③ 取一行值

如：
```
EXEC SQL FETCH c1 INTO :id, :name, :dept, :job, :years, :salary, :comm;
```

关于动态 FETCH 语句见 8.4 节。

④ 关闭游标

如：
```
EXEC SQL CLOSE c1;
```

关闭游标的同时，会释放由游标添加的锁和放弃未处理的数据。在关闭游标前，该游标必须已经声明和打开。另外，程序终止时，系统会自动关闭所有打开的游标。

【例 8.2】 编写嵌入式 SQL 语言程序，完成逐行打印 staff 表中的全部记录，显示的列

包括 id、name、dept、job、years、salary 和 comm 字段。

```
//声明游标
EXEC SQL DECLARE C1 CURSOR FOR
   SELECT id, name, dept, job, years, salary, comm FROM staff;
//打开游标
EXEC SQL OPEN c1; //打开游标
//成功获取到游标
while (SQLCODE = = 0)
{
   //取出游标的数据
    EXEC SQL FETCH c1 INTO :id, :name, :dept, :job, :years, :salary, :comm;
   //如果成功取出
    if (SQLCODE = = 0)
         printf(" % 4d % 12s % 10d % 10s % 2d % 8d % 8d",
                id, name, dept, job, years, salary, comm); //打印各个变量
}
//关闭游标
EXEC SQL CLOSE c1;
```

从上面的例子可以看出，首先需要定义游标结果集，即定义该游标的 SELECT 语句返回的行的集合。然后，使用 FETCH 语句逐行处理。值得注意的是，嵌入 SQL 语句中的游标定义选项同 Transact-SQL 中的游标定义选项有些不同，必须遵循嵌入 SQL 语句中的游标定义选项。

多行删除也可以使用 UPDATE 语句和 DELETE 语句来更新或删除由游标选择的当前行。使用 DELETE 语句删除当前游标所在的行数据的具体语法如下：

```
DELETE [FROM] {table_name | view_name} WHERE CURRENT OF cursor_name
```

其中：

- table_name 是表名，该表必须是 DECLARE CURSOR 中 SELECT 语句中的表。
- view_name 是视图名，该视图必须是 DECLARE CURSOR 中 SELECT 语句中的视图。
- cursor_name 是游标名。

【例 8.3】 编写嵌入式 SQL 语言程序，逐行显示 authors 表中的 firstname 和 lastname，并询问用户是否删除该信息，如果回答“是”，那么删除当前行的数据。

```
EXEC SQL DECLARE c1 CURSOR FOR
   SELECT au_fname, au_lname FROM authors;
EXEC SQL OPEN c1;
while (SQLCODE = = 0)
{
   EXEC SQL FETCH c1 INTO :fname, :lname;
   if (SQLCODE = = 0)
   {
        printf(" % 12s % 12s\n", fname, lname);
        printf("Delete? ");
        scanf(" % c", &reply);
```

```
            if (reply = = 'y')
            {
                EXEC SQL DELETE FROM authors WHERE CURRENT OF c1;
                printf("delete sqlcode =  % d\n", SQLCODE(ca));
            }
        }
    }
```

6. 错误处理

在每条嵌入式 SQL 语句之后立即编写一条检查 SQLCODE 值的程序，是一件很烦琐的事情。为了简化错误处理，可以使用 WHENEVER 语句。该语句是 SQL 预编译程序的指示语句，而不是可执行语句。它通知预编译程序在每条可执行嵌入式 SQL 语句之后自动生成错误处理程序，并指定了错误处理操作。

用户可以使用 WHENEVER 语句通知预编译程序去进行三种异常处理：

(1) WHENEVER SQLERROR action：表示一旦 SQL 语句执行时遇到错误信息，如 SQLCODE<0 则执行 action，action 中包含了处理错误的代码。

(2) WHENEVER SQLWARNING action：表示一旦 SQL 语句执行时遇到警告信息，则执行 action，即 action 中包含了处理警报的代码(SQLCODE=1)。

(3) WHENEVER NOT FOUND：表示一旦 SQL 语句执行时没有找到相应的元组，则执行 action，即 action 包含了处理没有查到内容的代码(SQLCODE=100)。

针对上述三种异常处理，用户可以指定预编译程序采取以下三种行为：

(1) WHENEVER …GOTO：通知预编译程序产生一条转移语句。

(2) WHENEVER…CONTINUE：通知预编译程序让程序的控制流转入到下一个主语言语句。

(3) WHENEVER…CALL：通知预编译程序调用函数。

其完整语法如下：

```
WHENEVER {SQLWARNING | SQLERROR | NOT FOUND} {CONTINUE | GOTO stmt_label | CALL function()}
```

【例 8.4】 编写嵌入式 SQL 语言程序，利用 WHENEVER 进行不同的错误处理。

```
EXEC SQL WHENEVER sqlerror GOTO errormessage1;
EXEC SQL DELETE FROM homesales
   WHERE equity < 10000;
EXEC SQL DELETE FROM customerlist
   WHERE salary < 40000;
EXEC SQL WHENEVER sqlerror CONTINUE;
EXEC SQL UPDATE homesales
   SET equity = equity - loanvalue;
EXEC SQL WHENEVER sqlerror GOTO errormessage2;
EXEC SQL INSERT INTO homesales (seller_name, sale_price)
   VALUES('Jane Doe', 180000.00); .
errormessage1:
   printf("SQL DELETE error: % ld\n, sqlcode);
exit();
```

```
errormessage2:
   printf("SQL INSERT error: %ld\n, sqlcode);
exit();
```

WHENEVER 语句使得对嵌入式 SQL 错误的处理更加简便。应该在应用程序中普遍使用,而不是直接检查 SQLCODE 的值。

8.1.4　动态 SQL 编程

前一节中讲述的嵌入 SQL 语言都是静态 SQL 语言,即在编译时已经确定了要引用的表和列。主变量不改变表和列信息。在上几节中,使用主变量改变查询参数,但是不能用主变量代替表名或列名。否则,系统报错。动态 SQL 语句就是来解决这个问题。

动态 SQL 语句的目的是,不是在编译时确定 SQL 的表和列,而是让程序在运行时提供,并将 SQL 语句文本传给 DBMS 执行。按照功能和处理上的划分,动态 SQL 应该分成两类来解释:动态修改和动态查询,即动态游标。

1. 动态修改

动态修改使用 PREPARE 语句和 EXECUTE 语句。PREPARE 语句是动态 SQL 语句独有的语句。其语法为:

```
PREPARE 语句名  FROM 主变量
```

该语句接收含有 SQL 语句串的主变量,并把该语句送到 DBMS。DBMS 编译语句并生成执行计划。在语句串中包含一个"?"表明参数,当执行语句时,DBMS 需要参数来替代这些"?"。PREPARE 执行的结果是,DBMS 把语句名赋给准备的语句。语句名类似于游标名,是一个 SQL 标识符。在执行 SQL 语句时,EXECUTE 语句后面是这个语句名。

【例 8.5】 编写嵌入式 SQL 语言程序,利用游标,向 books 表中插入数据。

```
EXEC SQL BEGIN DECLARE SECTION;
char       prep[] = "INSERT INTO books VALUES(?,?,?)";
char       bookname[30];
char       author[30];
int        num;
EXEC SQL END DECLARE SECTION;
EXEC SQL PREPARE prep_stat FROM :prep;
while (SQLCODE = = 0)
{
    strcpy(bookname, "C 编程");
    strcpy(author, "张三");
    num = 10;
    EXEC SQL EXECUTE prep_stat USING : bookname, : author, : num;
}
```

在例 8.5 中,prep_stat 是语句名,prep 主变量的值是一个 INSERT 语句,包含了三个参数(3 个"?")。PREPARE 的作用是:DBMS 编译这个语句并生成执行计划,并把语句名赋给这个准备的语句。

值得注意的是，PREPARE 中的语句名的作用范围为整个程序，所以不允许在同一个程序中将相同的语句名用在多个 PREPARE 语句中。EXECUTE 语句是动态 SQL 独有的语句。它的语法如下：

```
EXECUTE 语句名 USING 主变量 | DESCRIPTOR 描述符名
```

例 8.5 中的"EXEC SQL EXECUTE prep_stat USING ：bookname，：author，：num;"语句的作用是，请求 DBMS 执行 PREPARE 语句准备好的语句。当要执行的动态语句中包含一个或多个参数标志时，在 EXECUTE 语句中必须为每一个参数提供值，如：bookname、：author 和：num。这样的话，EXECUTE 语句用主变量值逐一代替准备语句中的参数标志（"?"），从而为动态执行语句提供了输入值。需要注意的是，USING 子句中的主变量数必须同动态语句中的参数标志数一致，而且每一个主变量的数据类型必须同相应参数所需的数据类型相一致。

2. 动态查询

游标分为静态游标和动态游标两类。对于静态游标，在定义游标时就已经确定了完整的 SELECT 语句。在 SELECT 语句中可以包含主变量来接收输入值。当执行游标的 OPEN 语句时，主变量的值被放入 SELECT 语句。在 OPEN 语句中，不用指定主变量，因为在 DECLARE CURSOR 语句中已经放置了主变量。

动态游标和静态游标不同。以下是动态游标使用的步骤。

【例 8.6】 编写嵌入式 SQL 语言程序，对 author 表的进行动态查询。

```
EXEC SQL BEGIN DECLARE SECTION;
char szLastName[] = "White";
char szFirstName[30];
EXEC SQL END DECLARE SECTION;
EXEC SQL
   DECLARE author_cursor CURSOR FOR
   SELECT au_fname FROM authors WHERE au_lname = :szLastName;
EXEC SQL OPEN author_cursor;
EXEC SQL FETCH author_cursor INTO :szFirstName;
```

动态游标和静态游标不同。以下是动态游标使用的过程：

① 声明游标：

对于动态游标，在 DECLARE CURSOR 语句中不包含 SELECT 语句，而是定义在 PREPARE 中的语句名，用 PREPARE 语句规定与查询相关的语句名称。

② 打开游标

完整语法为：

```
OPEN 游标名 [USING 主变量名 | DESCRIPTOR 描述名]
```

在动态游标中，OPEN 语句的作用是使 DBMS 在第一行查询结果前开始执行查询并定位相关的游标。当 OPEN 语句成功执行完毕后，游标处于打开状态，并为 FETCH 语句做准备。OPEN 语句执行一条由 PREPARE 语句预编译的语句。如果动态查询正文中包含有一个或多个参数标志时，OPEN 语句必须为这些参数提供参数值。USING 子句的作用是

规定参数值。

③ 取一行值

FETCH 语法为：

```
FETCH 游标名 USING DESCRIPTOR 描述符名
```

动态 FETCH 语句的作用是，把这一行的各列值送到 SQLDA 中，并把游标移到下一行注意而静态 FETCH 语句的作用是用主变量表接收查询到的列值。

在使用 FETCH 语句前，必须为数据区分配空间，SQLDATA 字段指向检索出的数据区。SQLLEN 字段是 SQLDATA 指向的数据区的长度。SQLIND 字段指出是否为 NULL。

④ 关闭游标

完整语法为：

```
EXEC SQL CLOSE 游标名
```

和关闭静态游标一样，也会释放由游标添加的锁和放弃未处理的数据。

在动态游标的 DECLARE CURSOR 语句中不包含 SELECT 语句，而是定义了在 PREPARE 中的语句名，用 PREPARE 语句规定与查询相关的语句名称。如果 PREPARE 语句中的语句包含了参数，那么在 OPEN 语句中必须指定提供参数值的主变量或 SQLDA。动态 DECLARE CURSOR 语句是 SQL 预编译程序中的一个命令，而不是可执行语句。该子句必须在 OPEN、FETCH、CLOSE 语句之前使用。

【例 8.7】 编写嵌入式 SQL 语言程序，完成 author 表的动态查询。

```
EXEC SQL BEGIN DECLARE SECTION;
char szCommand[] = "SELECT au_fname FROM authors WHERE au_lname = ?";
char szLastName[] = "White";
char szFirstName[30];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE author_cursor CURSOR FOR select_statement;
EXEC SQL PREPARE select_statement FROM :szCommand;
EXEC SQL OPEN author_cursor USING :szLastName;

EXEC SQL FETCH author_cursor INTO :szFirstName;
```

3. SQLDA

动态 SQL 语句在编译时可能不知道有多少列信息。在嵌入 SQL 语句中，这些不确定的数据是通过 SQLDA 完成的。SQLDA 的结构非常灵活，在该结构的固定部分，指明了多少列等信息，在该结构的后面有一个可变长的结构，说明每列的信息。在从数据库获得数据时，就可以采用 SQLDA 来获得每行的数据。各个数据库产品的 SQLDA 结构都不完全相同。可以通过 SQLDA 为嵌入 SQL 语句提供输入数据和从嵌入 SQL 语句中输出数据。理解 SQLDA 的结构是理解动态 SQL 的关键。关于 SQLDA 的具体内容请参看 SQL Server 的帮助。

8.2 存储过程

8.2.1 存储过程概述

1. 存储过程的概念和分类

存储过程是一组用来完成某种特定功能的 Transact-SQL 语句集合，这组 SQL 语句经过预编译后存储在数据库中，可以在 SQL Server 中或前端应用程序中对其进行调用。可以说存储过程是在数据库端执行的 Transact-SQL 程序，它主要用于实现需要频繁使用的查询。存储过程可以接收输入参数并以返回参数的形式向调用过程返回值。

在 SQL Server 中，存储过程主要分为两大类：系统存储过程和用户自定义存储过程。系统存储过程由 SQL Server 自动创建并存储在 master 数据库中，其名称都以 sp_为前缀。管理员可以通过 SQL Server 提供的系统存储过程进行管理性和信息性的工作，如查看数据库的相关信息、目录管理、配置和管理日志、安全性管理等，它们中的大部分可以在用户数据库中使用。用户自定义存储过程是由用户为完成某一特定功能所创建的存储过程，本章内容所涉及的主要是用户自定义的存储过程。

2. 存储过程的优点

在创建 SQL Server 应用程序时，Transact-SQL 是应用程序与 SQL Server 数据库之间主要的编程接口。在实际操作中，既可以创建本地存储的 Transact-SQL 程序，通过向 SQL Server 发送命令并处理结果，也可以将 Transact-SQL 程序作为存储过程存储在 SQL Server 中，并创建应用程序调用存储过程，再对数据结果进行相应处理。两种方法比较起来，后者具有一些显著的优点。

(1) 增强代码的可重用性和共享性

存储过程只需创建一次并存储在数据库中，就可以在程序中反复被调用，还可以被多个用户所共享。那些经常执行的查询操作可以写成存储过程，这样就避免了在程序中反复编写，提高了开发的效率和质量。

(2) 执行速度快

存储过程是预编译的，在第一次执行一个存储过程时，系统会对其进行分析和优化，并将经过编译的存储过程保存在高速缓存中，以后执行同一个存储过程时便无须再次进行编译，从而加快了其执行速度。

(3) 减少网络流量

存储过程存放在服务器端，客户端应用程序需要使用存储过程时，网络中只需传送对存储过程进行调用的语句，而无须传送大段的 Transact-SQL 代码，可以减少网络上的流量。

(4) 增强安全性

可以将执行存储过程的权限赋予某些用户而不将对数据表的直接访问权限授予他们，这样用户只能通过存储过程来访问和操作表中的数据，从而保证了数据的安全性。

8.2.2 创建和执行存储过程

可以用 CREATE PROCEDURE 语句和 EXECUTE 语句来创建和执行存储过程，也可以在 SQL Server 企业管理器中完成创建存储过程的操作。

创建存储过程的语法如下：

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ] ] [ ,...n ]
[ WITH
{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statements
```

以上语法中包含的主要参数说明如下：

procedure_name：所创建的存储过程的名称，存储过程的命名必须遵循标识符规则且在同一个数据库中必须唯一。

；number：可选整数，用来对同名的过程分组，以使用一条 DROP PROCEDURE 语句即可将同组过程删除。

@parameter：存储过程中的参数，在一个存储过程可以没有参数，也可以指定一个或多个参数。使用符号@作为第一个字符来指定参数名称，参数命名必须符合标识符的规则，而且符号@和参数名之间不能有空格。存储过程最多可以有 2100 个参数，在默认情况下，参数只能代替常量，而不能用于代替表名、列名或其他数据库对象的名称。

data_type：指定参数的数据类型。可以使用除 table 之外任何一种 SQL Server 所提供的数据类型来定义参数，同样也可以使用用户自定义类型。参数的数据类型确定了该参数所接受值的类型和范围，如果执行存储过程时所使用的值与参数的数据类型不兼容，就会出现错误。

VARYING：指定作为输出参数支持的结果集。

＝default：参数的默认值。如果定义了默认值，则在调用时不必指定该参数的值即可执行存储过程。默认值必须是常量或 NULL。如果对该参数使用 LIKE 关键字，那么默认值中可以包含通配符。

OUTPUT：表示参数为返回参数，使用 OUTPUT 参数可将信息返回给调用过程。

RECOMPILE：表示 SQL Server 不保存该存储过程的执行计划，每次执行都要重新编译。

ENCRYPTION：表示 SQL Server 加密 syscomments 表中包含 CREATE PROCEDURE 语句文本。存储过程一旦加密，其定义即无法解密，即使是存储过程的所有者或系统管理员也将无法查看存储过程定义。

FOR REPLICATION：指定不能在订阅服务器上执行为复制创建的存储过程。使用 FOR REPLICATION 选项创建的存储过程可用作存储过程筛选，且只能在复制过程中执行。本选项不能和 WITH RECOMPILE 选项一起使用。

AS：指定过程要执行的操作。

sql_statement：存储过程中包含的任意数目和类型的 Transact-SQL 语句。

在创建存储过程时还应该注意不能将 CREATE PROCEDURE 与其他 SQL 语句组合到一个批处理中。在 CREATE PROCEDURE 定义中不能出现几种 CREATE 语句，包括 CREATE DEFAULT、CREATE PROCEDURE、CREATE RULE、CREATE TRIGGER 和 CREATE VIEW，但可以在过程中创建其他的数据库对象。此外，只能在当前数据库中创建存储过程。

关于 CREATE PROCEDURE 的实际使用，将在后面的例子中进行详细说明。

执行存储过程使用 EXECUTE 语句，具体语法如下：

```
[ EXEC [ UTE ] ]
{
    [ @return_status = ]
    { procedure_name [ ;number ] | @procedure_name_var
}
[ [ @parameter = ] { value | @variable [ OUTPUT ] | [ DEFAULT ] ]
[ ,...n ]
[ WITH RECOMPILE ]
```

参数说明如下：

@return_status：一个可选的整型变量，保存存储过程的返回状态。

procedure_name：要进行调用的存储过程的名称。

@procedure_name_var：局部定义变量名，代表存储过程名称。

@parameter：存储过程参数，与前面在 CREATE PROCEDURE 语句中定义的相一致。

value：存储过程中参数的值。

@variable：用来保存参数或者返回参数的变量。

OUTPUT：指定存储过程必须返回一个参数。使用 OUTPUT 参数，目的是在调用存储过程的其他语句中使用其返回值，参数值必须作为变量传递。

下面通过几个简单的例子，对上述创建和执行存储过程的语法的使用进行说明。

【例 8.8】 创建存储过程，返回 Customers 表中所有客户的 CustomerID、CompanyName、Address 和 Phone 等信息。

```
CREATE PROCEDURE Customerinfo
AS
    SELECT CustomerID, CompanyName, Address, Phone FROM Customers;
GO
```

例 8.8 中没有任何参数，其中的 SQL 语句也非常简单，实际应用中可能很少会有如此简单的存储过程，但从中可以看到创建存储过程最基本的形式。存储过程中的 SQL 语句可以非常复杂，在设计的时候可以先单独编写和测试 SQL 语句，确定符合要求之后再按照存储过程的语法来进行定义。

要执行例 8.8 中的存储过程可采用以下语句：

```
EXEC Customerinfo;
```

【例 8.9】 创建带有输入参数的存储过程,实现返回某一客户曾经购买过的产品以及每种商品的数量。

```
CREATE PROCEDURE CustomerProducts @CustomerID nchar(5)
AS
   SELECT ProductName, Total = SUM(Quantity)
   FROM Products P, [Order Details] OD, Orders O, Customers C
   WHERE C.CustomerID  =  @CustomerID
        AND C. CustomerID  =  O. CustomerID AND O. OrderID  =  OD. OrderID AND OD. ProductID  =
P.ProductID
   GROUP BY ProductName
GO
```

执行这个存储过程时,需要将 CustomerID 作为参数给出:

```
EXEC CustomerProducts @CustomerID = 'ALFKI';
```

结果将返回 CustomerID 为 ALFKI 的客户所购买过的产品及数量。

【例 8.10】 编写存储过程,实现在 Customers 表中增加一个新的客户。

```
CREATE PROCEDURE AddNewCustomer
@CustomerID nchar (5),
@CompanyName nvarchar(40),
@ContactName nvarchar(30) = NULL,
@ContactTitle nvarchar(30)  = NULL,
@Address nvarchar(60)  = NULL,
@City nvarchar(15)  = NULL,
@Region nvarchar(15)  = NULL,
@PostalCode nvarchar(10)  = NULL,
@Country nvarchar(15)  = NULL,
@Phone nvarchar(24)  = NULL,
@Fax nvarchar(24)  = NULL
AS
INSERT INTO Customers
   (CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode,
Country, Phone, Fax)
VALUES
   (@CustomerID,@CompanyName,@ContactName,@ContactTitle, @Address, @City, @Region,
@PostalCode, @Country, @Phone, @Fax)
GO
```

在执行如本例所示的包含多个输入参数的存储过程时,可以有两种参数传递的方式:按位置传送和按参数名传送。按位置传送的方法在执行存储过程时,直接给出参数的值,值的顺序和创建存储过程的语句中参数定义的顺序一致,如下所示:

```
EXEC AddNewCustomer
   'GROST', 'GROSTLLA - Restaurant', 'Manuel Pereira','Owner',
   'Los Palos Grandes', 'Caracas', 'DF', '10784', 'Venezuela',
   '283 - 2951', '283 - 3397'
```

通过参数名传递的方法是在执行存储过程时给出参数的名称和参数值,如下所示:

```
EXEC AddNewCustomer
@CustomerID = 'GROST',
@CompanyName = 'GROSTLLA - Restaurant',
@ContactName = 'Manuel Pereira',
@ContactTitle = 'Owner',
@Address = 'Los Palos Grandes',
@City = 'Caracas',
@Region = 'DF',
@PostalCode = '10784',
@Country = 'Venezuela',
@Phone = '283 - 2951',
@Fax = '283 - 3397'
```

在这种方法中，多个参数可以按照任意的顺序给出，而无须与创建过程语句中的顺序一致，但如果有一个参数以这种方式给出，其他所有参数都必须按这种格式出现。

例 8.10 存储过程中除了 CustomerID 和 CompanyName 这两个参数外的其他参数都给出了默认值。如果指定了默认值，则在执行存储过程时可以不给出这些参数的值，需要注意的是，默认值必须是常量或 NULL。

【例 8.11】 创建带有输出参数的存储过程，返回某个供应商所供应的所有商品的品种数量。

```
CREATE PROCEDURE SupplierProduct
@CompanyName nvarchar(40), @ProductCount int OUTPUT
AS
    SELECT @ProductCount = COUNT( * )
    FROM Products P, Suppliers S
    WHERE
    S.SupplierID = P. SupplierID AND S. CompanyName  = @CompanyName
GO
```

关键字 OUTPUT 表明这是一个输出参数，需要注意，如果在一个存储过程定义中既有输入参数又有输出参数，那么输出参数应该位于所有输入参数之后。为了接收上述存储过程的返回值，在调用该过程的程序中必须声明用来接收输出参数的局部变量。此外，不能使用 OUTPUT 将常量传递给存储过程。

```
DECLARE @pCount int
EXEC SupplierProduct
'Karkki Oy', @pCount output
SELECT @pCount
GO
```

或者：

```
DECLARE @pCount int
EXEC SupplierProduct
@CompanyName  = 'Karkki Oy',
@ProductCount  = @pCount output
SELECT @pCount
GO
```

每个存储过程执行后都会返回一个状态值，调用程序可以根据这个状态值进行相应的处理。一般用 0 表示存储过程执行成功，用户也可以在存储过程中用 RETURN 语句返回自己指定的值。用输出参数也一样，如果用户要得到存储过程返回的状态值，需要事先声明一个接收变量，可以对例 8.10 的执行部分做简单修改：

```
DECLARE @status int
EXEC @status = AddNewCustomer
     @CustomerID = 'GROST',
     @CompanyName = 'GROSTLLA - Restaurant',
     @ContactName = 'Manuel Pereira',
     @ContactTitle = 'Owner',
     @Address = 'Los Palos Grandes',
     @City = 'Caracas',
     @Region = 'DF',
     @PostalCode = '10784',
     @Country = 'Venezuela',
     @Phone = '283 - 2951',
     @Fax = '283 - 3397'
IF @status = 0
     PRINT 'INSERT Operation Success!'
ELSE
     PRINT 'INSERT Operation Failed!'
GO
```

上例中，用变量@status 接收存储过程 AddNewCustomer 执行的返回值，并根据返回值显示相应的提示信息。

8.2.3 管理存储过程

1. 重新命名存储过程

可以使用 sp_rename 对现有存储过程重新命名，其格式为：

```
sp_rename 原过程名 新过程名
```

例如，要把例 8.11 中创建的存储过程 SupplierProduct 更名为 ProductOfSupplier，可以执行如下命令：

```
sp_rename SupplierProduct ProductOfSupplier
```

之后便可以使用新名称执行该存储过程了。在对象资源管理器中对存储过程进行重命名操作与在 Windows 资源管理器中对文件重命名一样，选择某个存储过程并右击，在弹出的菜单中选择“重命名”即可。

2. 修改存储过程

已经创建的存储过程中可能会存在一些逻辑上的错误，或者在使用过程中有了新的功能需求，这时可能需要对存储过程进行修改，修改存储过程可以通过 ALTER PROCEDURE 语句来实现，其语法如下：

```
ALTER PROC [ EDURE ] procedure_name [ ; number ]
[ { @parameter data_type }
    [ VARYING ] [ = default ] [ OUTPUT ]
] [ ,...n ]
[ WITH
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statements
```

其中各个参数和关键字的具体含义请参见 8.2.2 节中有关 CREATE PROCEDURE 语句的说明。

例 8.12 修改例 8.11 中的存储过程，实现返回某个供应商所供应的某类商品的品种数量，同时要求对过程进行加密。

【例 8.12】

```
ALTER PROCEDURE SupplierProduct
    @CompanyName nvarchar(40),
    @CategoryName nvarchar(15),
    @ProductCount int OUTPUT
WITH ENCRYPTION
AS
SELECT @ProductCount = COUNT( * )
    FROM Suppliers INNER JOIN
        Products ON Suppliers.SupplierID = Products.SupplierID INNER JOIN
        Categories ON Products.CategoryID = Categories.CategoryID
    AND Suppliers.CompanyName = @CompanyName
    AND Categories.CategoryName = @CategoryName
GO
```

从例 8.12 中可以看出，在修改存储过程的时候，既可以增加或删除一些如 WITH ENCRYPTION 这样的选项，也可以修改过程中 Transact-SQL 语句的内容。另外，在企业管理器中同样可以对存储过程的内容进行修改。

3. 删除存储过程

使用 DROP PROCEDURE 可以删除一个或多个存储过程，其语法为：

```
DROP PROCEDURE { procedure } [ ,...n ]
```

例如，要删除例 8.11 中创建的存储过程，可以执行以下命令：

```
DROP PROCEDURE SupplierProduct
```

8.2.4 系统存储过程

系统存储过程由 SQL Server 自动创建并存储在 master 数据库中，其名称都以 sp_为前缀。管理员可以通过 SQL Server 提供的系统存储过程进行管理性和信息性的工作，可以从其他数据库中执行系统存储过程，而无须使用 master 数据库名来完全限定该存储过程的名称。SQL Server 所提供的系统存储过程数量众多，可以对系统存储过程按类别进行分组。

一些常用的系统存储过程分类包括：

- 数据库维护计划过程：用于设置确保数据库性能所需的核心维护任务。
- 分布式查询过程：用于执行和管理分布式查询。
- 全文检索过程：用于执行和查询全文索引。
- 日志传送过程：用于配置和管理日志传送。
- 游标过程：执行游标变量功能。
- 安全过程：用于管理安全性。
- 复制过程：用于管理复制。
- SQL 事件探查器过程：由 SQL 事件探查器用于监视性能和活动。
- 系统过程：用于 SQL Server 的常规维护。

前文曾提到过的用于查看存储过程代码和相关信息的 sp_help 过程和 sp_helptext 过程就是属于“系统过程类”的存储过程。具体每一类别中所包括的系统过程以及它们的功能和使用方法，读者可参阅 SQL Server 联机丛书中的相关内容，此处不再赘述。

在创建自定义存储过程时，尽量要避免使用带有 sp_前缀的过程名称，因为 SQL Server 遇到以 sp_开头的存储过程首先会在 master 数据库中进行查找操作，之后再根据所提供的数据库名或所有者限定符查找存储过程，如果没有指定所有者，则以 dbo 为所有者进行查找。换句话说，如果当前数据库中存在带有 sp_前缀的用户自定义存储过程，即使使用数据库名对存储过程进行了限定，系统仍然会先检查 master 表，这在一定程度上会影响过程的执行效率。此外，还有更重要的一点，如果用户自定义的存储过程使用了和某个系统存储过程相同的名称，则用户过程将永远不会执行，因此在实际应用中一定要特别注意这些问题。

8.3 本章小结

嵌入式 SQL 的实现扩充宿主语言的编译程序，使之能执行 SQL 语句。SQL 和宿主语言的接口是数据库和宿主语言程序间信息的传递是通过共享变量实现的。嵌入式 SQL 编程分为静态 SQL 和动态 SQL 编程。静态 SQL 在编译时已经确定了引用的表和列。主变量不改变表和列信息。动态 SQL 不是在编译时确定 SQL 的表和列，而是让程序在运行时提供，并将 SQL 语句文本传给 DBMS 执行。按照功能和处理上的划分，动态 SQL 一般分成动态修改和动态查询，即动态游标。

在各种 DBMS 系统中，存储过程是非常重要的工具，是一组 Transact-SQL 语句集合。使用存储过程可以高效地完成对数据库的各种操作，而且可以多次使用，提高了程序的可重用性。熟练掌握存储过程并在数据库应用程序的开发中加以适当运用是非常有必要的。

第9章 数据库产品简介

自关系型数据库诞生以来，数据库产品层出不穷。本章将介绍主流数据库产品中 SQL Server、Oracle、MySQL、Sybase 和 DB2 的主要特点，并据此来选择合适的数据库产品。

9.1 SQL Server

SQL Server 是基于图形的管理界面、集中式的管理，支持多客户应用程序、企业级的应用程序以及多个不同的开发平台，运行于 Windows 操作系统上。SQL Server 是一个实际可运行的存储、维护和为应用系统提供数据的软件系统，是存储介质、处理对象和管理系统的集合体。它通常由软件、数据库和数据管理员组成。其软件主要包括操作系统、各种宿主语言，实用程序以及数据库管理系统。数据库是依照某种数据模型组织起来并存放在二级存储器中的数据集合。这些数据为多个应用服务，独立于具体的应用程序。数据库由数据库管理系统统一管理，数据的插入、修改和检索均要通过数据库管理系统进行。数据库管理系统是一种系统软件，它的主要功能是维护数据库并有效地访问数据库中任意部分数据，以保证在数据库维护过程中数据的完整性、一致性和安全性。

9.1.1 SQL Server 的简介

最早的 SQL Server 是由微软公司和 Sybase 公司联合开发的，1994 年 4 月以后双方决定各自独立开发自己的数据库系统。在短短几年时间里 SQL Server 得到了快速发展。从早期的 SQL Server 4.2 到现在的 SQL Server 2008，其间经过了几次大的升级，最重要的一次升级就是从 SQL Server 6.5 到 SQL Server 7.0 的变化，在 SQL Server 7.0 中，这个数据库核心部分经过重新设计，性能和可用性有了很大的提高，具备了从性能上挑战其他数据库产品的能力。2000 年，微软公司推出了企业级的数据库系统 SQL Server 2000。SQL Server 2000 很大一部分结构沿用了 SQL Server 7.0 的设计，在此基础上增加了许多有用的功能，增强了可靠性，并且提供了在线数据分析等商业化应用，引入了数据仓库、数据挖掘等新特性。SQL Server 2000 在市场上有很高的占有率，在当时成为占市场份额第一的数据库管理系统产品。SQL Server 在整个发展过程中一直是为运行 Windows 操作系统的 PC 服务器设计和优化的，在此期间 Windows 操作系统的 PC 服务器都有了巨大的进步，这也同时推动了 SQL Server 的发展。

2005 年，微软公司推出了经过重大改进的 SQL Server 2005。该版本开发周期长，在很

多地方对系统特性进行了改进。新引进了集成服务、报表服务等功能，增强了对.NET Framework 的支持，使基于数据库的应用开发效率和特性得到进一步的提升，奠定了大型企业级数据管理应用系统的基础。

2008 年，微软公司又推出了 SQL Server 2008。SQL Server 2008 是一个重大的产品版本，它推出了许多新的特性和关键的改进，使得它成为至今为止的最强大和最全面的 SQL Server 版本。

SQL Server 2008 在 Microsoft 的数据平台上发布，组织随时随地管理任何数据。它可以将结构化、半结构化和非结构化文档的数据(例如图像和音乐)直接存储到数据库中。SQL Server 2008 提供一系列丰富的集成服务，可以对数据进行查询、搜索、同步、报告和分析之类的操作。数据可以存储在各种设备上，从数据中心最大的服务器一直到桌面计算机和移动设备，都可以控制数据，而不用管数据存储在哪里。

SQL Server 2008 允许在使用 Microsoft .NET 和 Visual Studio 开发的自定义应用程序中使用数据，在面向服务的架构(SOA)和通过 Microsoft BizTalk Server 进行的业务流程中使用数据。信息工作人员可以通过他们日常使用的工具直接访问数据。SQL Server 2008 提供了一个可信的、高效率智能数据平台，可以满足用户的所有数据需求。

9.1.2　SQL Server 的特点

1. 客户机/服务器体系

客户机/服务器体系属于一种分布式计算技术，它的特点是程序中的所有数据处理过程，不像若干桌面系统的大型机那样发生在一台单独的计算机上。相反，SQL Server 的不同部分同时工作在不同的计算机上，彼此之间通过网络交换信息协调工作。SQL Server 运行在服务器上负责主要的数据处理工作。在客户端的计算机上客户端程序通过网络向服务器提交查询，并接收查询结果。

2. 关系型数据库

组织数据库中的数据有很多种方法，而关系数据库是其中最为高效的一种。关系型数据库具有以下特点：①关系模型的概念单一，实体和实体之间的联系用关系来表示；②以关系数学为基础；③数据的物理存储和存取路径对用户不透明；④关系数据库语言是非过程化的。

在关系型数据库中，数据被收集在表中，表包含了描述对数据库项目具有重要意义的对象的相关信息。每个表都由列和行组成。每一列的描述都代表对象的某个属性。每一行都表示表所属的对象的一个实例。

表和表之间存在着一些特定的关系，这些关系是由数据库设计者按照现实中的事物间的联系经过提取、抽象出来的，因此，关系型数据库在组织数据方面最接近人类的思维模式。

3. T-SQL 语言

处理关系型数据库最常用的是 SQL 语言，SQL 是结构化查询语言(Structured Query Language)的缩写，被美国国家标准化组织(ANSI)和国防标准化组织(ISO)采纳为关系型

数据库的标准开发语言，并制定了统一的标准 SQL Server 使用的 SQL 语言。微软公司在标准的 SQL 语言基础上加以扩充，称为 Transact-SQL（T-SQL），意思是面向事务处理的 SQL 语言。T-SQL 是 Microsoft SQL Server 应用程序使用的主要语言。

4. 支持数据复制

SQL Server 提供了完备的复制功能。通过使用复制功能，用户可以产生数据的复制件，移动这些数据的复制件到任何其他的地方，并自动化地同步数据，以便分布式环境下的多个数据复制件保持相同的数据值。SQL Server 的复制功能具有相当的灵活性，这使得在同一台服务器的数据库间或依赖网络互连的多台服务器间可以方便地实现复制。

复制工具 Replication Wizards 极大地简化了开发、实现和维护复制的步骤。其中，Configure Publishing and Distribution Wizard 被用于帮助用户将一台服务器标识为分发者，并可以选择标识其他的复制部件；Create Publication Wizard 可协助用户创建出版物；Push Subscription Wizard 可协助用户从一台服务器上将出版物“推”到一个或多个服务器甚至服务器组上；Pull Subscription Wizard 可协助用户从一台服务器上的出版物“拉”到其他服务器的数据库中；Disable Publishing and Distribution Wizard 可协助用户在一台服务器上禁止出版、分发；Replication Conflict Viewer 可协助用户查看和解决合并复制过程中发生的冲突。

5. 支持分布式事务处理

SQL Server 使用分布式事务处理协调程序（Distributed Transaction Coordinator，DTC）进行分布式事务处理（跨两种平台）。

6. 支持数据仓库

SQL Server 支持一些海量数据库的操作，这些数据库包含了来自于面向事务的数据库的数据。这些大型数据库用来研究趋势，这些趋势决非一般草率的检查可以发现的。而 SQL Server 所支持的数据仓库又是一个面向主题的、集成的、相对稳定的、反映历史变化的数据集合，用于支持管理决策。在支持决策方面，数据仓库采用面向分析型数据处理的方法，该方法使它不同于企业现有的操作型数据库；集成性方面，数据仓库是对多个异构的数据源的有效集成，集成后按照主题进行了重组，并包含了历史数据，而且这些存放在数据仓库中的历史数据往往不再修改。

7. 内建式的在线分析处理

SQL Server 的众多优点之一是将在线分析处理工具服务内建于数据库管理服务中。这部分服务称为微软决策支持服务（Microsoft Decision Support Services）。与市场上其他数据库产品所提供的服务不同，不用再购买一个通常很昂贵的第三方应用程序。这就大大降低了花在数据库使用者特别是数据库系统开发上的总费用。

9.1.3 SQL Server 2008 的新特性

SQL Server 2008 出现在微软数据平台愿景上是因为它使得公司可以运行他们最关键

任务的应用程序，同时降低了管理数据基础设施和发送观察信息给所有用户的成本。SQL Server 2008 在 SQL Server 2005 的基础上增加了对空间和非结构型数据的支持，并且增强了管理功能。

SQL Server 2008 具有以下特点：

- 可信任性——使得公司可以以很高的安全性、可靠性和可扩展性来运行他们最关键任务的应用程序。
- 高效性——使得公司可以降低开发和管理他们的数据基础设施的时间和成本。
- 智能性——提供一个全面的平台，可以在用户需要的时候给他发送信息。

1. 可信任性

在今天数据驱动的世界中，公司需要继续访问他们的数据。SQL Server 2008 为关键任务应用程序提供了强大的安全特性、可靠性和可扩展性。

(1) 信息保护

在过去的 SQL Server 2005 的基础之上，SQL Server 2008 做了以下方面的增强来扩展它的安全性：简单的数据加密、外键管理和审计。

① 简单的数据加密

SQL Server 2008 可以对整个数据库、数据文件和日志文件进行加密，而不需要改动应用程序。在 SQL Server 2008 里，有几个加密选择。其一是透明数据加密，整个数据库可以通过 SQL 引擎加密。该方式加密了所有数据库的数据和日志文件。进行加密使公司可以满足遵守规范及其关注数据隐私的要求。第二个加密方式是备份加密。SQL Server 2008 备份加密的方式可以防止数据泄露和被篡改。简单的数据加密的好处包括使用任何范围或模糊查询搜索加密的数据、加强数据安全性以防止未授权的用户访问。

② 外键管理

SQL Server 2008 为加密和密钥管理提供了一个全面的解决方案。为了满足不断发展的对数据中心信息安全性更强的需求，公司投资给供应商来管理公司内的安全密钥。如果要处理信用卡或遵循 PCI 的处理，SQL Server 2008 将支持硬件安全模块(HSM)。HSM 是在独立于要保护的数据的本地用来存储密钥的第三方硬件解决方案。

③ 审计

SQL Server 2008 允许监控数据的更改或访问，从而提高了遵从性和安全性。SQL Server 2008 具有像服务器中加强的审查的配置和管理这样的功能，这使得公司可以满足各种规范需求。SQL Server 2008 还可以定义每一个数据库的审查规范，所以审查配置可以为每一个数据库做单独的制定。

(2) 加强应用程序稳定性

SQL Server 2008 继续使公司具有提供简化了管理并具高可靠性的应用的能力。

① SQL Server 2008 基于 SQL Server 2005，并提供了更可靠的加强了数据库镜像的平台。新的特性包括页面自动修复、提高了性能和加强了可支持性。

② 热添加 CPU，为了在线添加内存资源而扩展 SQL Server 中的已有的支持，热添加 CPU 使数据库可以按需扩展。事实上，CPU 资源可以添加到 SQL Server 2008 所在的硬件平台上而不会影响相关程序的正常运行。

(3) 系统性能最佳化与预测功能

SQL Server 2008 提供了一个广泛的功能集合，使数据平台上的所有工作负载的执行都是可扩展的和可预测的。SQL Server 2008 推出了范围更大的数据采集，一个用于存储性能数据的新的集中的数据库，以及新的报表和监控工具。在备份压缩和数据压缩等方面，SQL Server 2008 也有显著的性能改进。SQL Server 2008 通过提供了一个新的制定查询计划的功能，从而提供了更好的查询执行稳定性和可预测性。

2. 高效性

SQL Server 2008 降低了管理系统、. NET 架构和 Visual Studio Team System 的时间和成本，使得开发人员可以开发强大的下一代数据库应用程序。

(1) 基于政策的管理

作为微软正在努力降低公司的总成本所做的工作的一部分，SQL Server 2008 推出了陈述式管理架构(DMF)，它是一个用于 SQL Server 数据库引擎的新的基于策略的管理框架。这种陈述式管理为用户提供了以下优点：遵从系统配置的政策；监控和防止通过创建不符合配置的政策来改变系统的操作；简化管理工作以便达到减少公司总成本的目的；使用 SQL Server 管理套件查找遵从性问题。

陈述式管理架构是一个基于政策的系统，用于管理一个或多个 SQL Server 2008 实例。DMF 由三个组件组成：政策管理、创建政策的政策管理员和显式管理。政策管理员选择一个或多个要管理的对象，并显式检查这些对象是否遵守指定的政策，或显式地使这些对象遵守某个政策。

(2) 安装配置的改进

SQL Server 2008 对 SQL Server 的服务生命周期提供了显著的改进，它重新设计了安装、建立和配置架构。这些改进将计算机上的各个安装与 SQL Server 软件的配置分离开来，这使得公司和软件合作伙伴可以提供推荐的安装配置。

(3) 加速开发过程

SQL Server 提供了集成的开发环境和更高级的数据提取，使开发人员可以创建下一代数据应用程序，同时简化了对数据的访问。改进主要包括 ADO. NET 实体框架、语言集成查询能力、CLR 集成和 ADO. NET 对象服务、Service Broker 可扩展性和 Transact-SQL 的改进。

(4) 创建偶尔连接系统

SQL Server 2008 推出了一个统一的同步平台，可以在应用程序、数据存储和数据类型之间达到一致性同步。在与 Visual Studio 合作的基础上，SQL Server 2008 可以通过 ADO. NET 中提供的新的同步服务和 Visual Studio 中的脱机设计器快速地创建偶尔连接系统。偶尔连接成为一种新的工作方式。SQL Server 2008 提供了支持，使得可以改变跟踪和使客户以最小的执行消耗进行功能强大的执行，以此来开发基于缓存的、基于同步的和基于通知的应用程序。

(5) 新增数据类型

应用程序正在结合使用越来越多的数据类型，而不仅仅是过去数据库所支持的那些。SQL Server 2008 基于过去对关系数据的强大支持，提供了新的数据类型使得开发人员和

管理员可以有效地存储和管理非结构化数据,例如文件、文档和图片。还增加了对管理高级地理数据的支持。除了新的数据类型,SQL Server 2008 还提供了一系列对不同数据类型的服务,提供一个丰富的服务集合来进行数据交互作用,同时为数据平台提供了可靠性、安全性和易管理性。

3. 智能性

商业智能继续作为大多数公司投资的关键领域和对于公司所有层面的用户来说的一个无价的信息源。SQL Server 2008 提供了一个全面的平台,可以为用户提供用户所需要的智能化服务。

(1) 集成大量数据

公司继续投资于商业智能和数据仓库解决方案,以便从他们的数据中获取商业价值。SQL Server 2008 提供了一个全面的和可扩展的数据仓库平台,它可以用一个单独的分析存储进行强大的分析,以满足成千上万的用户在几兆字节的数据中的需求。

(2) 发送相应的报表

SQL Server 2008 提供了一个可扩展的商业智能基础设施,使得 IT 人员可以在整个公司内使用商业智能来管理报表以及任何规模和复杂度的分析。SQL Server 2008 报表服务提供另一个完全基于服务器的平台,它被设计用于支持广泛的报表需求,使得公司可以有效地以用户想要的格式和他们的地址发送相应的、个人的报表给成千上万的用户。SQL Server 2008 报表服务提供了制作从很多数据源获得数据并具有丰富的格式的报表所需要的工具和功能,并提供了一组全面的工具用来管理和保护企业报表解决方案。这使得用户可以在他们各自的领域对相关信息进行及时访问,使得他们可以做出更好、更快、更符合实际的决策。

(3) 使用户获得全面的洞察力

在线分析处理的前提条件是能及时访问到准确的信息,使用户在应对问题,甚至非常复杂的问题时能做出快速反应。SQL Server 2008 在 SQL Server 2005 强大的 OLAP 能力的基础上做了进一步改进,为所有用户提供了更快捷的查询速度。这个性能的提升使得公司可以执行具有许多维度和聚合的非常复杂的分析。这个执行速度与 Microsoft Office 的深度集成相结合,使 SQL Server 2008 可以让所有用户获得全面的洞察力。

9.1.4 应用程序访问 SQL Server 的实例

以下是 C# 访问示例数据库的表 Reader 的一个小例子。

```
//连接数据库
String Cnstr = @"server = .;Initial Catalog = DATA;Integrated Security = True";
SqlConnection con = new SqlConnection(Cnstr);
//使用 DataSet 读取数据
String strSQL = "SELECT Rno,Rname,Rank,Sex,Age FROM Reader";
SqlDataAdapter da = new SqlDataAdapter(strSQL, con);
DataSet ds = new DataSet();
da.Fill(ds, "Reader");
//显示数据
```

```
String s = "";
Foreach(DataRow dr in ds.Tables[0].Rows)
{ s = s + "Rno:" + dr["Rno"].ToString() + " Rname:" + dr["Rname"].ToString() + " Rank:" + dr
["Rank"].ToString() + " Sex:" + dr["Sex"].ToString() + " Age:" + dr["Age"].T oString() + " ";
}
Label1.Text = s;
```

9.2 Oracle

Oracle 数据库系统是美国 Oracle 公司(甲骨文)提供的以分布式数据库为核心的一组软件产品,是目前流行的客户/服务器(CLIENT/SERVER)或 B/S 体系结构的数据库之一。它支持真正的对象的概念,应用在中大型系统中,主要用于 Linux、UNIX 系统上,现在也有了运行于 Windows NT 系统上的产品。

9.2.1 Oracle 的发展历程

1977 年,Larry Ellison、Bob Miner 和 Ed Oates 成立了 Relational Software Incorporated (RSI)公司,他们使用 C 和 SQL 接口开发了关系数据库关系系统——Oracle。不久,他们推出了版本 1 为一个原型。在 1979 年,他们发行了第一个产品。

Oracle RDBMS 版本 2 开始是工作在 Digital PDP-11 机器(运行 RSX-11 操作系统)上的,后来工作在 DESVAX 系统上。

1983 年推出了版本 3,它几乎全部由 C 语言编写而成,RSI 也改名为 Oracle 公司。

1984 年推出了版本 4,该版本支持 VAX 系统和 IBM VA 操作系统。

1985 年推出了版本 5,使用 SQL * NET,从此引入了客户机/服务器计算,因此成为一个新的里程碑。它也是第一个突破 640KB 限制的 MS-DOS 产品。

1989 年推出了版本 6,引入了低层锁,并有许多性能改善和功能增强。此时,Oracle 可以运行在许多平台和操作系统上。1991 年,在 DEC VAX 平台上的 Oracle RDBMS 版本 6.1 中引入了 Oracle Parallel Server 选项(OPS),不久并行服务器选项可运行于多种平台。

1992 年 Oracle 7 诞生,它采用了多线索服务器体系结构,能够在所有硬件体系结构上为大量用户提供可扩充性等功能。Oracle 7 在内存 CPU 和工作使用方面进行了许多结构性修改。

1997 年推出了 Oracle 8,它增强了对象扩展和许多新特征及管理工具。Oracle 8 是一个紧密集成的对象关系数据库管理系统方案,它没有像其他数据库产品那样只是在关系数据库和用户端应用软件之间提供一个对象服务器网关,或者在现有的数据库上附加一个采用对象技术的外壳。关系型数据库和对象技术的结合,使用户不需要移植现有 Oracle 7 应用软件,便能够在 Oracle 8 上使用,极大保护了现有客户的投资。

1999 年又推出了 Oracle 8i,作为世界上第一个全面支持 Internet 的数据库,Oracle 8i 是唯一一个具有集成模式 Web 信息管理工具的数据库,也是世界上第一个具有内置 Java 引擎的可扩展的企业级数据库平台。它具有在一个易于管理的服务器中同时支持数个用户的能力,可以帮助企业充分利用 Java 以满足其迅速增长的 Internet 应用需求。通过支持

Web高级应用所需要的多种媒体数据来支持Web繁忙站点不断增长的负载需求，Oracle 8i提供了在Internet上运行电子商务所必需的可靠性、可扩展性、安全性和易用性。

2000年发布了Oracle 9i，继承和改进了8i的特征。

2003年9月发布了Oracle 10g。10g支持网格计算，即多台结点服务器利用高速网络组成一个虚拟的高性能服务器，负载在整个网格中衡，按需增删结点，避免了单点故障。Oracle 10g支持自动管理增删硬盘，根据需要自动分配和释放系统内存，可以快速纠正人为错误的闪回查询和恢复，可以恢复数据库、表甚至记录。相对于Oracle 9i而言，Oracle 10g存储数据的表空间可以跨平台复制，极大地提高了数据仓库加载速度，容灾的数据卫士增加了逻辑备份功能，备份数据库日常可以运行于只读状态，充分利用了备份数据库。

2007年7月发布了Oracle 11g。Oracle从出现伊始发展到现在的Oracle 11g，成为第一款为网格计算而设计的数据库。Oracle 11g集成了Oracle数据库管理技术的所有优势，又融入了网格计算的各种新的性能特点。

9.2.2 Oracle的特点

1. 支持多用户、大事务量的高性能的事务处理

引入了共享SQL和多线索服务器体系结构。这些技术的引入增强了Oracle的能力，并大大减少了Oracle的资源占用，使之在低档软硬件平台上用较少的资源就可以支持更多的用户，而在高档平台上则可以支持成百上千个用户。

2. 数据安全性和完整性控制

Oracle数据库提供了基于角色分工的安全保密管理，它在数据库管理的功能、安全性、完整性的检查、一致性方面都有突出的表现。

3. 提供对数据库操作的接口

Oracle提供对数据库操作的接口，遵守数据存取语言、操作系统、用户接口和网络通信协议的工业标准。

4. 支持分布式数据库和分布处理

Oracle数据库自从Oracle 5起就提供了分布式处理能力，到Oracle 7就有比较完善的分布式数据库功能了，可以让客户通过网络比较方便地读写远端数据库里的数据，并提供对称复制的技术。

5. 具有可移植性、可兼容性和可连接性

(1) 兼容性

Oracle产品采用标准SQL，并经过美国国家标准技术所(NIST)测试。与IBM SQL/DS、DB2、INGRES、IDMS/R等兼容。

(2) 可移植性

Oracle的产品可运行于很宽范围的硬件与操作系统平台上。可以安装在70种以上不

同的大、中、小型机上，可在 VMS、DOS、UNIX、Windows 等多种操作系统下工作。

(3) 可连接性

能与多种通信网络相连，支持各种协议(如 TCP/IP、DECnet、LU6.2 等)。

6. 完整的数据管理功能

Oracle 数据管理功能完备，具有数据的大量性、数据保存的持久性、数据的共享性和数据的可靠性。

7. 完备关系的产品

Oracle 是完备关系的产品，遵循以下准则：

- 信息准则——关系型 DBMS 的所有信息都应在逻辑上用一种方法，即表中的值显式地表示；
- 保证访问的准则；
- 视图更新准则：只要形成视图的表中的数据变化了，相应视图中的数据也同时发生变化；
- 数据物理性和逻辑性独立准则。

8. 轻松地实现数据仓库的操作

数据仓库需要从各种不同的数据源取得各种不同的数据，并且把这些巨大数据量的数据转换成对于用户可用的数据，为企业的决策提供数据支持。这个过程常常被称为 ETL，即提取、转换、装载。Oracle 9i 引进了新的"边装载、边转换"的办法来取代过时的串行处理步骤：先转换然后装载或者先装载然后转换。在 Oracle 这种新方法里，数据库参与了数据转换和装载的过程，成为了 ETL 过程的一个有机组成部分。而另外有些原来必需的步骤则没有继续存在的必要了，另一些则可以得到改进。Oracle 9i 也因这种功能便利的数据仓库的操作而变得快速高效。

9. 支持大量多媒体数据

Oracle 支持大量多媒体数据，如二进制图形、声音、动画以及多维数据结构等。

9.2.3 Oracle 的开发工具

Oracle 产品主要包括数据库服务器、开发工具和连接产品三类，并且提供了多种开发工具，能极大地方便用户进行进一步的开发。

Oracle 提供的开发工具包是 Developer、Designer、Discover、Oracle Office 等，它涵盖了从建模、分析、设计到具体实现的各个环节。

(1) Developer

它包括：Oracle Forms 用于快速生成基于屏幕的复杂应用，具有 GUI 界面和多媒体功能，主要用于操纵数据和查询；Oracle Reports 是快速生成报表的工具，能生成各种复杂的报表，同样能处理多媒体信息；Oracle Graphics 用于生成各种图形应用；Oracle Books 用于生成联机文档。

(2) Designer

这是 Oracle 提供的 CASE 工具。该工具能够帮助用户对复杂系统进行建模、分析和设计。还可以帮助用户绘制 E-R 图、功能分层图、数据流图和方阵图。

(3) Discover

这是一个 OLAP 工具,主要用于支持数据仓库应用。它可以对历史数据进行挖掘,以找到发展趋势,对不同层次的概况数据进行分析,以便发现有关业务的详细信息。

(4) Oracle Office

适用于办公自动化,能完成企业范围内的消息接收与发送、日程安排、日历管理、目录管理以及拼写检查。

9.2.4 应用程序访问 Oracle 的实例

以下是 Jsp 连接 Oracle 9i/10g 的实例:

```
<%
//使用 thin 模式连接数据库
String url = "jdbc:Oracle:thin:@localhost:1521:test";
// localhost 为数据库 ip 地址,1521 为连接端口号,test 为数据库名
String user = "scott";
String password = "pass";
Connection conn =  DriverManager.getConnection(url,user,password);
//查询数据
Statement stmt = conn.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT Rno,Rname,Rank,Sex,Age FROM Reader";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()) { %>
Rno 为: <% = rs.getString(1) %>
Rname 为: <% = rs.getString(2) %>
Rank 为: <% = rs.getString(3) %>
Sex 为: <% = rs.getString(4) %>
Age 为: <% = rs.getString(5) %>
<% } %>
<% out.print("数据库操作成功."); %>
//关闭连接
<%
rs.close();
stmt.close();
conn.close();
 %>
```

9.3 MySQL

MySQL 是一个真正的多用户、多线程 SQL 数据库服务器。MySQL 是以一个客户机/服务器结构实现的,是由一个服务器守护程序和很多不同的客户程序和库组成的。相对于

其他系统而言，MySQL 数据库可以称得上是目前运行速度最快的 SQL 语言数据库。可运行在大多数的 Linux 平台，以及少许非 Linux 甚至非 UNIX 平台上。通俗一点说，MySQL 是一个精巧的 SQL 数据库管理系统，虽然它不是开放源代码的产品，但在某些情况下可以自由使用。由于它的强大功能、灵活性、丰富的应用编程接口(API)以及精巧的系统结构，受到了广大自由软件爱好者甚至是商业软件用户的青睐，特别是与 Apache 和 PHP/PERL 结合，为建立基于数据库的动态网站提供了强大的动力。在与 PHP 的配合使用中被 Linux 下的 Web 开发者称为 PHP 的“黄金搭档”。

9.3.1 MySQL 简介

MySQL 是一个小型关系型数据库管理系统，开发者为瑞典 MySQL AB 公司。该公司于 2008 年 1 月 16 日被 Sun 公司收购。而 2009 年，Sun 又被 Oracle 收购。目前 MySQL 被广泛地应用在 Internet 上的中小型网站中。由于其体积小、速度快、总体拥有成本低，尤其是具有开放源码这一特点，许多中小型网站为了降低网站总体拥有成本而选择 MySQL 作为网站数据库。

MySQL 包含一些与 SQL 标准不同的转变，它们大多数被设计成对 SQL 脚本语言不足的一种补偿。然而，另一些扩展确实使 MySQL 与众不同。例如，LINK 子句搜索是自动忽略大小写的。MySQL 也允许用户自定义的 SQL 函数，函数必须被编译到一个共享库文件中(.so 文件)，然后用一个 LOAD FUNCTION 命令装载。它也缺乏一些常用的 SQL 功能，没有子选择(在查询中的查询)，视图也没有了。当然大多数子查询可以用简单的连接(join)子句重写，但有时用两个嵌套的查询思考问题比一个大连接容易。

MySQL 的主要的缺陷之一是缺乏标准的 RI 机制，解决办法之一是支持唯一索引。Rule 限制的缺乏(在给定字段域上的一种固定的范围限制)通过大量的数据类型来补偿。不简单地提供检查约束、外部关键字和经常与 RI 相关的“级联删除”功能。MySQL 是数据库领域的中间派。它缺乏一个全功能数据库的大多数主要特征，但是又有比类似 Xbase 记录存储引擎更多的特征。它像企业级 RDBMS 那样需要一个积极的服务者守护程序，但是不能像他们那样消费资源。查询语言允许复杂的连接查询，但是所有的参考完整都必须由程序员强制保证。

9.3.2 MySQL 的特点

1. 多线程

多线程是 MySQL 的关键特性，MySQL 的核心程序采用完全的多线程编程。线程是轻量级的进程，它可以灵活地为用户提供服务，而不过多地占用系统资源。用多线程和 C 语言实现的 MySQL 能很容易地充分利用 CPU，可以采用多 CPU 体系结构。

2. 开放性

MySQL 是自由的开放源代码产品，它所使用的语言 SQL 以 ANSI SQL2 为基础，这个数据库引擎可运行在多个平台上，包括 Windows 2000、MacOSX、Linux、FreeBSO 和 Solaris，如果没有可用于平台的二进制文件，则可将源代码在相应的平台上进行编译。

3. 多用户支持

同时访问数据库的用户数量不受限制。MySQL 可有效地满足 50～1000 个并发用户的访问,并且在超过 600 个用户限度的情况下,MySQL 的性能没有明显地下降。

4. 用户权限设置简单、有效

MySQL 有一个非常灵活而且安全的权限和口令系统。当客户与 MySQL 服务器连接时,二者之间所有的口令传送被加密,而且 MySQL 支持主机认证。

5. MySQL 可运行在不同的操作系统下

简单地说,MySQL 可以支持 Windows 95/98/NT/2000 及以上版本,以及 UNIX、Linux 和 SUN OS 等多种操作系统平台。这意味着在一个操作系统中实现的应用可以很方便地移植到其他的操作系统下。

6. MySQL 支持 ODBC for Windows

MySQL 支持所有的 ODBC 2.5 函数和其他许多函数,这样就可以用 Access 连接 MySQL 服务器,从而使得 MySQL 的应用被大大扩展了。

7. MySQL 支持大型的数据库

虽然对于用 PHP 编写的网页来说,只要能够存放上百条以上的记录数据就足够了,但 MySQL 可以方便地支持上千万条记录的数据库。作为一个开放源代码的数据库,MySQL 可以针对不同的应用进行相应的修改。

8. 性能高效稳定

MySQL 拥有一个非常快速而且稳定的基于线程的内存分配系统,可以持续使用而不必担心其稳定性。事实上,MySQL 的稳定性足以应付一个超大规模的数据库。

9. 强大的查询功能

MySQL 支持查询的 SELECT 和 WHERE 语句的全部运算符和函数,并且可以在同一查询中混用来自不同数据库的表,从而使得查询变得快捷和方便。

10. 应用程序支持

PHP 为 MySQL 提供了强力支持,PHP 中提供了一整套的 MySQL 函数,对 MySQL 进行了全方位的支持。

11. 为多种编程语言提供了 API

MySQL 为多种编程语言提供了 API。这些编程语言包括 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby 和 Tcl 等。

9.3.3 MySQL 的开发工具

使用各种精心设计的工具来管理 MySQL 数据库要比单纯使用传统方法轻松得多。开发人员应该不断寻找那些能够缩短开发时间的工具。

1. MySQL Workbench

MySQL Workbench 是一个由 MySQL 开发的跨平台、可视化数据库工具。它作为 DBDesigner4 工程的替代应用程序而备受瞩目。

2. phpMyAdmin

phpMyAdmin 是一款免费的、用 PHP 编写的工具，用于在万维网上管理 MySQL，它支持 MySQL 的大部分功能。phpMyAdmin 能够支持管理数据库、表格、字段、联系、索引、用户、许可等一些最常用的操作，同时还可以直接执行任何 SQL 语句。

3. Aqua Data Studio

对于数据库管理人员、软件开发人员以及业务分析师来说，Aqua Data Studio 是一个完整的集成开发环境(IDE)。它主要包括数据库查询和管理工具，数据库、源代码管理以及文件系统的比较工具，为 SVN 和 CVS 设计了一个完整的集成源代码管理客户端，提供了一个强大的数据库建模工具。

4. SQLyog

SQLyog 是一个全面的 MySQL 数据库管理工具。它包含了开发人员在使用 MySQL 时所需的绝大部分功能，包括查询结果集合、查询分析器、服务器消息、表格数据、表格信息，以及查询历史，它们都以标签的形式显示在界面上，开发人员只需操作鼠标即可完成任务。此外，它还可以方便地创建视图和存储过程。

5. mytop

mytop 是一款基于控制台的工具，用于监视线程以及 MySQL 3.22.x、3.23.x 和 4.x 服务器的整体性能。mytop 的灵感来自系统监视工具 top。mytop 连接到 MySQL 服务器之后，能定期运行 SHOW PROCESSLIST 和 SHOW STATUS 命令，并以一种有用的格式总结从这些命令中所获得的信息。

6. MySQL Sidu

MySQL Sidu 是一款免费的 MySQL 客户端，它通过网络浏览器来运行，这款工具简单易学。Sidu 这几个字母表示 Select(选择)、Insert(插入)、Delete(删除)和 Update(更新)。

7. Navicat Lite MySQL Admin Tool

Navicat 是一款非常快速、可靠的数据库管理工具，受到广大用户的普遍欢迎。Navicat 专门用来简化数据库管理并且减少管理成本，它旨在满足数据库管理人员、数据库开发人员

以及广大中小企业的需要。Navicat 有一个非常直观的 GUI,可以使用户安全便捷地创建、组织、访问以及分享信息。

对于 MySQL 来说,Navicat 工具是一个强大的数据库管理和开发工具。它可以跟任何版本的 MySQL 数据库服务器一起工作,并且支持 MySQL 大多数最新的功能,包括 Trigger、Stored Procedure、Function、Event, View 和 Manage User 等。Navicat Lite 可以免费下载,但是仅适用于非商业活动。

9.4 Sybase

Sybase SQL Server 是由 Sybase 公司开发的具有良好性能、高可靠性、面向联机事务处理的可编程的关系型数据库服务器。Sybase 主要有三种版本:一是 UNIX 操作系统下运行的版本,二是 Novell Netware 环境下运行的版本,三是 Windows NT 环境下运行的版本。对于 UNIX 操作系统,目前广泛应用的为 Sybase 10 及 Syabse 11 for SCO UNIX。

9.4.1 Sybase 数据库的发展史

1984 年,Mark B. Hiffman 和 Robert Epstern 创建了 Sybase 公司,推出了支持企业范围的"客户/服务器体系结构"的数据库。吸取了 INGRES 的研制经验,以满足联机事务处理应用的要求,于 1987 年推出了 Sybase SQL Server,称为大学版 INGRES 的第三代产品。Sybase System 12.5 是其最新产品,支持企业内部各种数据库应用需求,如数据仓库、联机事务处理、决策支持系统和小平台应用等。Sybase 的产品系列囊括了数据库、开发工具、集成中间件、企业门户,以及移动无线服务器等。Sybase 的企业软件解决方案适用于所有的技术平台,极大地保证了产品的兼容性,真正实现了 Everything Works Better When Everything Works Together。

9.4.2 Sybase 数据库的特点

1. 基于客户/服务器体系结构的数据库

一般的关系数据库都是基于主/从式的模型。在主/从式的结构中,所有的应用都运行在一台机器上。用户只是通过终端发命令或简单地查看应用运行的结果,而在客户/服务器结构中,应用被分在了多台机器上运行。一台机器是另一个系统的客户,或是另外一些机器的服务器。这些机器通过局域网或广域网连接起来,好处是支持共享资源且在多台设备间平衡负载,允许容纳多个主机的环境,充分利用了企业已有的各种系统。

2. 真正开放的数据库

由于采用了客户/服务器结构,应用被分在了多台机器上运行。更进一步,运行在客户端的应用不必是 Sybase 公司的产品。对于一般的关系数据库,为了让其他语言编写的应用能够访问数据库,提供了预编译。Sybase 数据库,不只是简单地提供了预编译,而且公开了应用程序接口 DB-LIB,鼓励第三方编写 DB-LIB 接口。由于开放的客户 DB-LIB 允许在不

同的平台使用完全相同的调用，因而使得访问 DB-LIB 的应用程序很容易从一个平台向另一个平台移植。

3. 一种高性能的数据库

Sybase 真正吸引人的地方还是它的高性能，具体体现在以下几方面。

（1）可编程数据库

通过提供存储过程，创建了一个可编程数据库。存储过程允许用户编写自己的数据库子例程。这些子例程是经过预编译的，因此不必为每次调用都进行编译、优化、生成查询规划，因而查询速度要快得多。

（2）事件驱动的触发器

触发器是一种特殊的存储过程。通过触发器可以启动另一个存储过程，从而确保数据库的完整性。

（3）多线索化

Sybase 数据库体系结构的另一个创新之处就是多线索化。一般的数据库都依靠操作系统来管理与数据库的连接。当有多个用户连接时，系统的性能会大幅度下降。Sybase 数据库不让操作系统来管理进程，把与数据库的连接当作自己的一部分来管理。此外，Sybase 的数据库引擎还代替操作系统来管理一部分硬件资源，如端口、内存、硬盘，绕过了操作系统这一环节，提高了性能。

4. 扩充性能

数据库服务器可以实现从只有少数用户访问的数据库到成千上万用户连接的服务器集成系统。由于其数据库服务器的吞吐量和响应时间都是可以预测的，因此为用户有计划地扩充硬件设备提供了很好的支持。

5. 高可用性

Sybase 的数据库服务器的高可用性表现在它支持每天 24 小时、每周 7 天的不间断运行。同时，数据库所具有的联机高速备份和恢复机制将用户由于系统故障而引起的损失减少到最小程度。

6. 数据完整性

Sybase 的数据完整性体现在以下方面：首先，它通过允许用户使用存储过程和触发器来实现对用户数据的完整性控制；另外，它允许对表、列、视图、存储过程、命令等对象进行授权，用户可以根据需要设定指定的用户或组是否具有编辑和运行权限。

7. 互操作性

数据库的互操作性主要表现在以下几个方面：

（1）支持多种互联标准。

（2）灵活的事务语义提供、ANSI/ISO 事务模式选项、支持 ODBC 和 X/A 标准、支持多网络协议、支持分布式数据库管理。

(3) 面向开发人员支持可编程的两个阶段提交。开发人员可以设置错误处理方式，而不仅局限于系统缺省方式。

8. 管理方便

Sybase的事务日志的备份非常灵活，可按照规定的时间间隔进行备份，这样用户可以随时根据需要恢复任何一个时刻的数据库原型。另一方面，其提供的远程管理方式由于采用的是中央集中式对远程结点进行管理，因此用户可以直接集中对远程多个服务器进行性能参数的调整。

9.4.3 Sybase数据库的组成

Sybase数据库主要由三部分组成：

(1) 进行数据库管理和维护的一个联机的关系数据库管理系统Sybase SQL Server：Sybase SQL Server是个可编程的数据库管理系统，它是整个Sybase产品的核心软件，起着数据管理、高速缓冲管理、事务管理的作用。

(2) 支持数据库应用系统的建立与开发的一组前端工具Sybase SQL Toolset：ISQL是与SQL Server进行交互的一种SQL句法分析器。ISQL接收用户发出的SQL语言，将其发送给SQL Server，并将结果以形式化的方式显示在用户的标准输出上。

DWB是数据工作台，是Sybase SQL Toolset的一个主要组成部分，它的作用在于使用户能够设置和管理SQL Server上的数据库，并且为用户提供一种对数据库的信息执行添加、更新和检索等操作的简便方法。在DWB中能完成ISQL的所有功能，且由于DWB是基于窗口和菜单的，因此操作比ISQL简单，是一种方便实用的数据库管理工具。

APT是Sybase客户软件部分的主要产品之一，也是从事实际应用开发的主要环境。APT工作台是用于建立应用程序的工具集，可以创建从非常简单到非常复杂的应用程序，它主要用于开发基于表格(Form)的应用。其用户界面采用窗口和菜单驱动方式，通过一系列的选择完成表格(Form)、菜单和处理的开发。

(3) 可在异构环境下把其他厂商的应用软件和任何类型的数据连接在一起的接口Sybase Open Client/Open Server：通过Open Client的DB-LIB库，应用程序可以访问SQL Server。而通过Open Server的SERVER-LIB，应用程序可以访问其他的数据库管理系统。

9.4.4 Sybase数据库的开发工具

Sybase为用户提供了良好的开发工具和开发环境，支持组件创建和快速应用开发。

1. PowerBuilder

PowerBuilder是一个可视化的客户/服务器应用开发工具，其强大的功能可以帮助用户快速开发复杂的应用，并且它还提供与其他数据库的接口。

2. Power Designer

Power Designer是一套紧密集成的计算机辅助软件工程(CASE)工具，用于为复杂的

数据库应用完成分析、设计、维护、建立文档和创建数据库等功能。

3. Power J

Power J 是基于 Java 应用的快速开发工具。

4. Powre++

Powre++是一组 RAD C++客户/服务器和 Internet 面向对象的开发工具。

9.5 DB2

DB2 是 IBM 公司研制的一种关系型数据库系统。DB2 主要应用于大型应用系统，具有较好的可伸缩性，可支持从大型机到单用户环境，应用于 OS/2、Windows 等平台。DB2 提供了高层次的数据利用性、完整性、安全性、可恢复性，以及小规模到大规模应用程序的执行能力，具有与平台无关的基本功能和 SQL 命令。DB2 通用数据库主要组件包括数据库引擎应用程序接口和一组工具。数据库引擎提供了关系数据库管理系统的基本功能，包括管理数据、控制数据的访问、保证数据完整性及数据安全等。所有数据访问都通过 SQL 接口进行。

9.5.1 DB2 的发展历程

DB2 是 IBM 公司研制的一种关系型数据库管理系统，分别在不同的操作系统平台上服务。从最初的研究到今天的 DB2 9，DB2 这个数据库领域里的佼佼者走过了很长的一段路。1983 年，IBM 发布了 DATABASE 2(DB2) for MVS(内部代号为 Eagle)。1987 年 IBM 发布了 OS/2 V1.0 扩展版，这是 IBM 第一次把关系型数据库处理能力扩展到微机系统，这也是 DB2 for OS/2、UNIX 和 Window 的雏形。1992 年，第一届 IDUG 欧洲大会在瑞士日内瓦召开，这一重大事件标志着 DB2 应用的全球化。1993 年，IBM 发布了 DB2 for OS/2 V1 和 DB2 for RS/6000 V1，这是 DB2 第一次在 Intel 和 UNIX 平台上出现。之后的研究在不断地进行，该产品也在不断地改进和完善。2006 年 IBM 发布 DB2 9，将数据库领域带入 XML 时代。XML 语言以其可扩展性、与平台无关性和层次结构等特性，成为构建 SOA (Service-Oriented Architecture)时不同应用间进行数据交换的主流语言。DB2 9 这款新品最大的特点就是率先实现了 XML 和关系数据间的无缝交互，而无须考虑数据的格式、平台或位置。IBM DB2 9 是 IBM 在数据库领域多年开发项目最重要的成果，它将传统的静态数据库技术转变为交互式的动态数据服务器，使用户能够更好地管理文档、视频和音频文件、图片、网页等所有类型的信息。可以说，DB2 9 开创了一个新的 XML 数据库时代。

DB2 有多种不同的版本，如 DB2 工作组版(DB2Workgroup Edition)、DB2 企业版(DB2 Enterprise Edition)、DB2 个人版(DB2 Personal Edition)和 DB2 企业扩展版(DB2 Enterprise-Expended Edition)等，这些产品基本的数据管理功能是一样的，它们之间的区别在于支持远程客户能力和分布式处理能力上的不同。个人版适合于单机使用，即服务器只能由本地应用程序访问。工作组版提供了本地和远程客户访问 DB2 的功能，当然远程客户

要安装相应的客户应用程序开发部件。企业版不仅包括工作组版中的所有部件，而且还增加了对主机连接的支持。企业扩展版则允许将一个大的数据库分布到同一类型的多个不同计算机上，这种分布式功能尤其适用于大型数据库的处理。

9.5.2 DB2 的特点

DB2 数据库提供了高层次的数据利用性、完整性、安全性、可恢复性，以及小规模到大规模应用程序的执行能力，具有与平台无关的基本功能和 SQL 命令。

1. 支持面向对象的编程

DB2 支持复杂的数据结构，如无结构文本对象，可以对无结构文本对象进行布尔匹配、最接近匹配和任意匹配等搜索。可以建立用户数据类型和用户自定义函数。

2. 支持多媒体应用程序

DB2 支持大二分对象，允许在数据库中存取二进制大对象和文本大对象。其中，二进制大对象可以用来存储多媒体对象。

3. 备份和恢复能力

重新启动中断的恢复操作，可以在数据库恢复时节省宝贵的时间，同时简化了恢复工作。支持重定向恢复操作，在现有备份镜像中自动生成脚本。能够从表空间备份镜像中重新构建数据库。此项功能让 DB2 的恢复更加灵活和多样化，同时也为客户提供了更全面的恢复解决方案。

4. 查询优化

它以拥有一个非常完备的查询优化器而著称。其外部连接改善了查询性能，并支持多任务并行查询。

5. 分布式数据库访问

DB2 具有很好的网络支持能力，每个子系统都可以连接十几万个分布式用户，可同时激活上千个活动线程，对大型分布式应用系统尤为适用。

6. 支持数据复制

9.5.3 DB2 的开发工具

IBM 提供了许多开发工具，主要有 visualizer、visualage、visualgen 等几种。

1. visualizer

这是一种客户/服务器环境中的集成工具软件，主要包括 visualizer query 可视化查询工具、visualizer ultimedia query 可视化多媒体查询工具、visualizer chart 可视化图标工具、

visualizer procedure 可视化过程工具、visualizer statistics 可视化统计工具、visualizer plans 可视化规划工具、visualizer development 可视化开发工具。

2. visualage

它是一个功能很强的可视化的面向对象的应用开发工具，可以大幅度地提高软件开发效率。其主要特征有：

- 可视化程序设计工具。
- 部件库。
- 关系数据库支持。
- 群体程序设计。
- 支持增强的动态连接库。
- 支持多媒体。
- 支持数据共享。

3. visualgen

visualgen 是 IBM 所提供的高效开发方案的重要组成部分。它集成了第四代语言、客户/服务器与面向对象技术，给用户提供了一个完整、高效的开发环境。

9.5.4 应用程序访问 DB2 的实例

以下是 C# 访问示例数据库的表 Reader 的一个小例子。

```
//连接数据库
/创建 DB2 的数据源连接
OdbcConnection con = new OdbcConnection ( " Dsn = DataSourceName; uid = UserName; pwd =
mismidas");
//使用 DataSet 读取数据
String strSQL = "SELECT Rno,Rname,Rank,Sex,Age FROM Reader";
OdbcDataAdapter da = new OdbcDataAdapter(strSQL, con);
DataSet ds = new DataSet();
da.Fill(ds, "Reader");
//显示数据
String s = "";
Foreach(DataRow dr in ds.Tables[0].Rows)
{ s = s + " Rno:" + dr["Rno"].ToString() + " Rname:" + dr["Rname"].ToString() + " Rank:" + dr
  ["Rank"].ToString() + " Sex:" + dr["Sex"].ToString() + " Age:" + dr["Age"].ToString() + " ";
}
Label1.Text = s;
```

9.6 本章小结

数据库技术的发展，已经成为先进信息技术发展环节中一个相当重要的组成部分，是现

代计算机信息系统和计算机应用系统的基础和核心。随着科学技术的发展，计算机技术不断应用到各行各业，数据存储不断膨胀的需要也会对未来的数据库技术提出更高的要求。面对令人眼花缭乱的数据库产品市场，应当根据个人或企业的不同需求，有针对性地选择适合自己的数据库产品。本章介绍了主流数据库产品中 SQL Server、Oracle、MySQL、Sybase 和 DB2 的主要特点，以便用户在选择数据库产品时有所侧重。

第10章 数据库技术的新发展

数据库技术从20世纪60年代中期产生到今天仅有不到50年的历史，但其发展速度之快、使用范围之广是其他技术望尘莫及的。短短40多年间，已从第一代的层次和网状数据库系统发展到支持关系数据模型的关系数据库系统，我们称之为第二代数据库系统，再进一步发展为第三代以面向对象模型为主要特征的新一代数据库系统。第三代数据库系统保持或继承了第二代数据库系统的技术，除提供传统的数据管理服务外，第三代数据库系统支持更加丰富的对象结构和规则，集数据管理、对象管理和知识管理于一体。同时，数据库技术与多学科技术在发展的过程中有机结合，计算机领域中其他新兴技术的发展对数据库技术产生了重大影响。传统的数据库技术与网络通信技术、人工智能技术、面向对象程序设计技术、并行计算技术等互相渗透，互相结合，成为当前数据库技术发展的一个主要特征，随之也涌现出了各种新型的数据库系统。另一方面，数据库技术被应用到特定的领域。面向应用领域的数据库技术的研究在传统数据库系统基础上，结合各个应用领域的特点，研究适合该应用领域的数据库技术，如数据仓库、工程数据库、统计数据库、科学数据库、空间数据库、地理数据库等，这是当前数据库技术发展的又一重要特征。

10.1 面向对象的数据库系统

面向对象的数据库是基于面向对象数据模型的新一代数据库系统。把数据库技术与面向对象程序设计方法相结合形成了面向对象的数据库系统（Object-Oriented database system，OODBS）。

传统的层次、网状和关系数据库在许多商业数据库应用中取得了极大成功，然而在设计和实现更为复杂的数据库应用时，传统数据库系统暴露出了一些缺陷。在设计与实现科学实验数据库、电信数据库、地理信息系统数据库以及多媒体数据库的时候出现了新的应用要求，如长事务的处理、图像或大文本项等新数据类型的存储以及非标准的特殊应用操作。传统的数据库操作往往不能满足这种复杂数据库应用的要求。

面向对象程序设计方法已经被广泛地应用于软件工程、知识库、人工智能和计算机系统等领域。面向对象程序设计方法和数据库技术的结合，不但能让设计者定义复杂对象的结构，还能让设计者定义作用于这些复杂对象的操作，从而能够有效地支持新一代数据库的应用。

10.1.1 面向对象数据库系统的基本特征

数据库的特征依赖于实际应用,所设计的数据库语言必须允许用户方便地使用这些特征,数据库的结构也应能有效地支持这些特征。面向对象数据库系统区别于传统数据库系统的主要特征如下。

1. 具有表达和管理对象的能力

面向对象数据库系统应该具有表达和管理对象的能力。面向对象数据库系统通过对象及它们之间的相互联系来描述现实世界。它应该支持对象标识,使得对象的存在不依赖于本身的值,而只依赖于它的标识,对象间能够通过对象标识而相互区分。类的层次和继承是一个关键的概念,新的类允许从以前定义过的类那里继承结构和操作。因此在面向对象数据库系统中,新的对象类型可以简便地重用已有的类型定义。面向对象数据库系统的一个问题是如何表示对象间的联系。ODMG 2.0(Object Database Management Group,对象数据库管理组)标准中提出用一对反向引用来表示二元关系,即把与某个对象相关的对象的标识放在对象内部,并维护参照完整性。

2. 具有任意复杂度的对象结构

面向对象数据库系统中的对象可以具有任意复杂度的对象结构。这使对象能够包含所有描述该对象的必要信息。传统数据库恰好与此相反,它把关于复杂对象的信息分散在许多关系或记录中,从而丧失了现实世界的对象与数据库表示之间的直接对应关系。在面向对象数据库系统中,允许逐步细化复杂实体,还能将整个复杂对象或其子集作为一个独立的单位,可以在某一时刻将某一成员对象加进去。

3. 具有与面向对象编程语言交互的接口

面向对象数据库系统必须具有与面向对象编程语言交互的接口。面向对象程序设计中的对象是瞬态对象,只在程序执行过程中存在。而面向对象数据库可以延长对象的存在,把对象持久地存储起来。对象在程序结束之后仍然持续存在,可以被检索或被其他程序使用。面向对象数据库系统通过与面向对象编程语言交互的接口,可以提供持久化对象和共享对象的能力,从而允许多个程序和应用共享这些对象。

4. 具有表达和管理数据库变化的能力

面向对象数据库应具有表达和管理数据库变化的能力。管理同一对象的多个版本的能力对于设计和工程应用而言是至关重要的。一个对象的旧版本代表一个已经通过测试和鉴定的设计方案,那么应该保存这个版本直到新版本通过测试和鉴定。除了允许版本变化外,面向对象数据库系统也应该允许模式演变。所谓模式演变是指类的声明发生了变化,或创建了新的类或联系。

可见,一个面向对象数据库系统首先应是一个数据库系统,同时又必须具有面向对象的特征。

10.1.2 面向对象数据模型

面向对象数据模型(Object-Oriented data model,OO 模型)是一种可扩充的数据模型,它以对象(实体)为基础,综合了语义数据模型、知识表达数据模型中的一些基本概念,借鉴了面向对象程序设计和抽象数据模型的一些基本思想,支持面向对象分析和设计。

目前,虽然面向对象数据模型还没有一个统一的标准,但基本包含以下核心概念。

1. 对象

E-R 模型中的实体在 OO 模型中都是对象(Object)。对象是由一组数据结构和在这组数据结构上的操作的程序代码封装起来的基本单位。

对象描述包含静态和动态两方面的内容,即对象的结构和行为。在 OO 模型中,对象的结构和行为分别用属性和方法来表示。

(1) 属性

一个对象包含若干个属性(Attribute),用以描述对象的状态、组成和特征。这些属性的值就构成该对象的静态描述。需要注意的是,在面向对象模型中,属性可能也是对象,例如一个计算机由主机、显示器、键盘等组成,显示器、键盘等本身也是对象。因此,一个属性可以包含其他的对象,这样引用对象的过程可以递归下去,从而组成各种复杂的对象。一个对象可以被多个对象引用,这种由对象组成对象的过程称为聚集(Aggregation)。

(2) 方法

对象除了具有静态特征之外还具有行为特征。例如,一个学生对象不仅包含学号、姓名、性别、身高、体重等描述静态特征的属性,而且还具有上课、考试等行为特征。因此除了属性之外,对象还包括若干方法(Method),用以描述对象的行为特征。方法可以改变对象的状态,对对象进行各种操作。方法的定义包含了方法的接口和方法的实现。

2. 对象标识

每个对象都有一个在系统内唯一的、不变的标识,称做对象标识(Object Identifier)。所有的对象都具有唯一标识并且是可区分的,这个"唯一"指的是对象可由其内在本质区分,而不是通过它们的描述性质来区分。对象标识独立于对象的内容,一般由系统产生,用户不能修改。对象标识的值对外部用户来说是不可见的,但系统会在内部用这个值唯一地标识每个对象,并用这个值创建和管理内部对象引用。

3. 封装和消息

封装(Encapsulation)是 OO 模型中的一个关键的概念。每一个对象都是其属性和方法的封装,封装的目的在于两方面:①把方法的调用接口和实现分开,实现细节对使用是隐藏的。这样实现部分的修改可不致影响接口,有利于数据的独立性。②对象封装后,直接受对象中所定义的操作。对象状态的改变只能通过自身的方法来进行。外界程序不能直接访问对象的属性,可以避免许多不希望发生的副作用,有利于提高程序的可靠性。

外界与对象的通信一般只能借助于消息(Message)。消息是用来请求对象执行某一处理或回答某些信息的要求。消息传递给对象,调用对象的相应的方法进行相应的操作,再以

消息的形式返回操作结果。消息传递是对象联系的唯一方法。

4. 类

具有相同的属性和方法的所有对象构成一个对象类(Class),通常称做类。类中的每个对象称做该类的一个实例。例如,如果把学生定义为一个类,那么某个学生就是学生类中的一个对象。同一类的对象所具有的相同的属性和方法,这些属性和方法只需要在类中定义一次,而不必在每个对象中重复定义。任何对象都属于某一特定的类。

类的概念类似于关系模式,类的属性类似于关系模式中的属性,对象类似于元组。如果把类本身看做一个对象,则称之为类对象。与类对象相关的属性集和方法集适用于该类对象而不适用于该类实例,这样的属性和方法称为类属性、类方法。类属性是描述类的所有对象的共同特征的属性,对于任何对象实例,它的属性值是相同的。例如学生类的属性“学生人数”,虽然它和每个学生有关系,但并不是某个学生的属性,而是学生类的属性,在同一时刻,对于每个学生来说,“学生人数”是完全相同的。面向对象数据库模式是类的集合。

5. 继承

OO 模型可以直接表示 is-a 关系。如果 A is-a B,则称类 A 是类 B 的子类(Subclass),称类 B 是类 A 的超类(Superclass)。面向对象模型的这种超类和子类间的关系可以形成一个类层次结构,如图 10-1 所示。

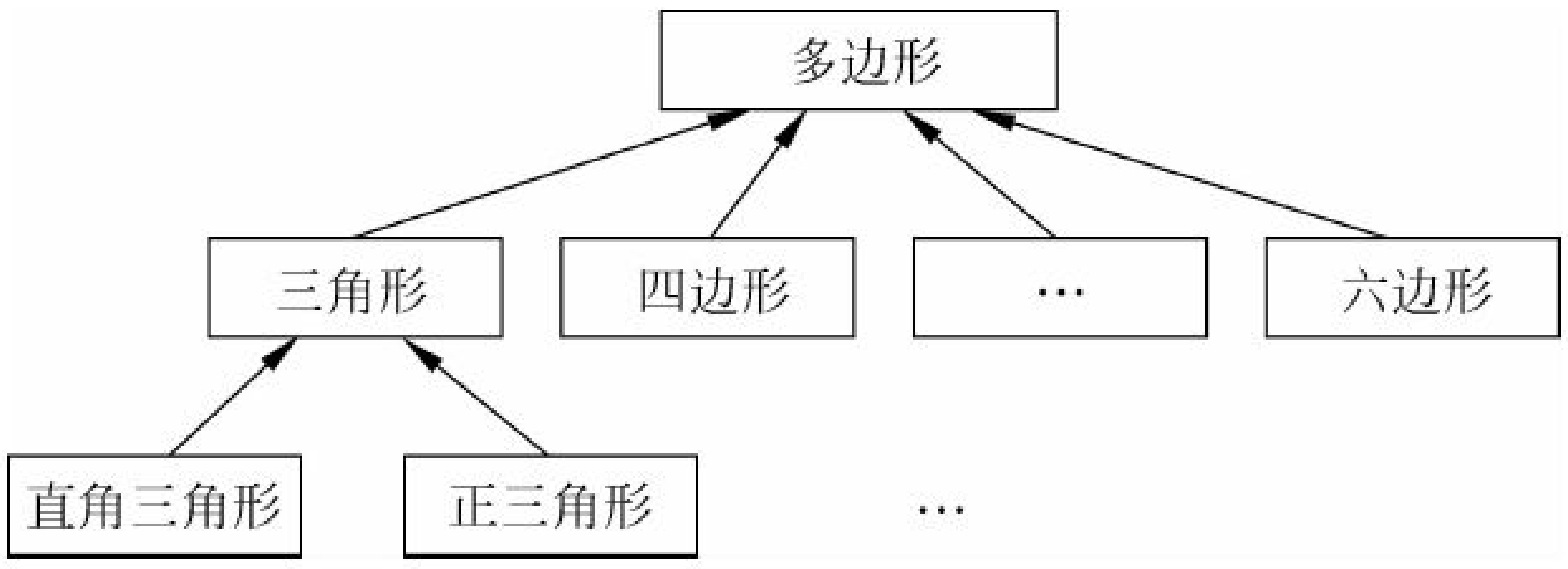

图 10-1　类层次结构示例

从概念上说,类层次结构从上至下是一个特殊化的过程,反之,自下而上是一个普遍化的过程。图 10-1 所表达的类层次结构表明直角三角形和正三角形都是三角形,而三角形、四边形和六边形等都是多边形。

在面向对象模型中,常用的有单继承与多重继承两种继承(Inheritance)。若一个子类只能继承一个超类的特性,这种继承称为单继承;若一个子类能继承多个超类的特性,这种继承称为多重继承(Multiple Inheritance)。

类的继承带来了很多好处,子类在继承父类特性的同时,还可以定义自身的属性、方法,并可剔除掉父类中不适合子类的操作。由封装和继承还可导出面向对象的其他优良特性,如多态性、动态联编等。

10.1.3　面向对象数据库语言

面向对象数据库语言用于描述面向对象数据库的模式,说明并操纵类定义与对象实例。与关系数据库的标准语言 SQL 类似,面向对象数据库语言主要包括对象定义语言和对象操

纵语言。对象查询语言是对象操纵语言的一个重要子集。

面向对象数据库语言一般应具备下列功能：

(1) 类的定义和操纵。面向对象数据库语言可以操纵类，包括定义、生成、存取、修改与撤销类。其中类的定义包括定义类的属性、操作特征、继承性与约束等。

(2) 操作/方法的定义。面向对象数据库语言可用于对象操作/方法的定义与实现。在操作实现中，语言的命令可用于操作对象的局部数据结构。对象模型中的封装性允许操作/方法由不同程序设计语言来实现，并且隐藏不同程序设计语言实现的事实。

(3) 对象的操纵。面向对象数据库语言可以用于操纵实例对象。

对象数据库管理组织(Object Database Management Group，ODMG)是面向对象数据库管理系统软件商的国际联盟，成立于1991年，并于1993年发布第一版标准，即ODMG-93或者ODMG 1.0。但当时外界对于该标准的评价并不高，很多人甚至认为它根本不能称为一个"标准"，至多只能称为一个关于面向对象数据库语言的"提议"集合。但此后ODMG广泛采纳各方面对于标准的意见，快速推出了多个改进版本。发展到ODMG 3.0时已经被广泛认可为一个比较成熟的关于面向对象数据库的标准了。ODMG标准是在面向对象数据库和对象关系数据库逐渐取代纯关系型数据库成为市场和研究的新宠的背景下诞生的。它规范和引导对象数据库市场，以解决由于缺少标准而造成各种对象数据库产品间缺乏可移植性的问题。ODMG对象模型是从OMC对象模型继承演变而来的，它是ODMG标准的核心。ODMG对象模型主要提供了数据类型、类型构造器以及其他一些可以用对象定义语言来说明对象数据库模式的概念，它是对象定义语言和对象查询语言的基础。

对象定义语言用来定义符合对象模型的对象类型。值得重视的是，它是ODMG实现可移植性的主要方法。对象定义语言被设计为支持ODMG对象模型的语义结构，并且独立于任何特定的编程语言。它的主要用途是创建对象说明，也就是类和接口。因此对象定义语言不是一种完全的编程语言。用户可以独立于任何编程语言在对象定义语言中指定一种数据库模式，然后使用特定的语言绑定来指明如何将对象定义语言结构映射到特定编程语言的结构，如C++、SMALLTALK和Java。

对象查询语言是专门为ODMG对象模型指定的查询语言。对象查询语言被设计为能与编程语言紧密配合使用。这些编程语言有一个ODMG绑定的定义，如C++、SMALLTALK和Java。这样，嵌入某种编程语言的一个对象查询语言的查询，可以返回与那种语言的类型系统相匹配的对象。对于查询而言，对象查询语言的语法和关系型标准查询语言SQL的语法相似，只是增加了有关对象的特征，如对象标识、复杂对象、操作、继承和多态性。

面向对象数据库语言是面向对象数据库区别于传统数据库的一个重要特征。但面向对象数据库语言的查询功能很弱，这是因为用变量引用对象的方式不能对对象进行统一管理。例如，图书作为类，作者是属性。面向对象数据库语言可从每一本书的对象找到它的所有作者，但不支持另一方向的查询，即从作者查询他的所有作品，除非把作者也定义成一个类，同时把作品设置成属性，这样的设计将在数据库中造成冗余，为此就有必要扩充面向对象数据库语言的查询功能。

数据库系统从网状模型到关系模型的进步，使数据库查询语言从用户导航式的过程性语言进入到了由系统自动选择查询路径的非过程性语言。但是非过程性语言的面向集合的操作方式又与高级程序设计语言的面向单个数据的操作方式之间产生了不协调现象，俗称

"阻抗失配"。阻抗失配的根本原因在于关系数据库的数据模型和程序设计语言不一致。因而，对嵌入式数据库语言来说，不可避免地会产生阻抗失配。但面向对象数据库不同，它的数据模型概念来自面向对象的程序设计方法，因此作为某一面向对象的程序设计语言扩充的面向对象数据库语言，能够从根本上解决阻抗失配问题。

商业的关系型数据库管理系统成功的一个原因在于 SQL 标准。对象数据库管理组织提出的 ODMG 1.0 到 ODMG 3.0 标准包含了对象定义语言和对象操纵语言。但是目前还没有一个关于面向对象数据语言的标准能够像 SQL 标准那样得到业界的普遍支持。ODMG 标准还有待于扩展和完善，比如用视图概念扩展 ODMG 等。标准的建立是在尝试和探索中前进的。作为一个开放性的组织，需要新的公司和学术研究团体不断加入进来，才能保持 ODMG 旺盛的生命力。

10.1.4 对象关系数据库

在今天的商业领域中，有许多可用的数据库管理系统产品，占统治地位的主要有两个：关系数据库系统和面向对象数据库系统，分别支持关系数据模型和对象数据模型。数据库管理系统产品的另外两种主要类型是层次数据库和网状数据库，它们分别基于层次和网状数据模型。随着数据库技术的发展，后两种数据库系统会逐渐被前两种所取代。

数据库系统面临着许多领域新的应用的挑战，如音频和视频处理系统中的数字化信息、计算机辅助桌面排版系统中的大文本、人造卫星成像或天气预报中的图像、股票市场交易历史中的时间序列数据，以及地图数据中的空间和地理数据。显然需要设计某些数据库，它们可以开发、操纵和维护来自这些应用的复杂对象。

在面对上述复杂应用时，基本关系模型及其 SQL 语言的早期版本被证明是不适用的。层次数据模型可以很好地适用于组织自然中存在的分层结构，但它在数据中的内置层次路径上过于局限和固定；网状数据模型可以明确地对联系建模，但在实现方面却需要使用大量的指针，而且不具备对象标识、继承和封装这类概念，也不支持多种数据类型和复杂对象。因此产生了一种趋势——将对象数据模型中的特征和语言结合到关系数据模型中，这样就扩展了关系数据模型，形成了对象关系数据库系统，使它能够处理当今复杂的应用。

对象关系数据模型扩展关系数据模型的方式是提供一个包括复杂数据类型和面向对象的更丰富的类型系统。关系查询语言也需要做相应的扩展以处理这些更丰富的类型系统。对象关系数据库系统以对象关系数据模型为基础，为想要使用面向对象特征的关系数据库用户提供了一个方便的迁移途径。

1. 嵌套关系

关系数据理论中定义了第一范式，它要求所有的属性都具有原子域。原子域指这个域中的元素是不可再分的单元。然而并非所有的应用都用第一范式关系建模最好。例如，某些应用的用户将数据库视为一个对象的集合，而不是一个记录的集合，这些对象可能需要用数条记录来表示。一个简单易用的界面要求用户直观概念上的一个对象与数据库系统概念上的一个数据项之间是一一对应的关系。

嵌套关系模型是关系模型的一个扩展，域可以是原子的也可以赋值为关系。元组在一个属性上的取值可以是一个关系，于是关系可以存储在关系中，从而形成了关系的嵌套。这

样，一个复杂的对象就可以用嵌套关系的单个元组来表示。如果将嵌套关系的一个元组视为一个数据项，在数据项和用户数据库观念上的对象之间就有了一一对应的关系。

2. 复杂类型

嵌套关系只是基本关系模型扩展的一个实例，其他非原子数据类型，如嵌套记录，同样已被证明是有用的。面向对象数据模型已经导致了对对象的继承和引用等特征的需求。有了复杂对象系统和面向对象，就能够直接表达E-R模型的一些概念，如实体标识、多值属性、一般化和特殊化，而不再需要经过关系模型的复杂转化。

3. 继承、引用类型

这里的介绍是基于SQL-99标准的，不过也会提到一些在这个标准中没有出现但在SQL标准的未来版本中会引入的一些特征。

继承可以在类型的级别上进行，也可以在表的级别上进行。首先考虑类型的继承，再假定需要在数据库中对那些是学生或教师的人分别存储一些额外的信息。

SQL-99只支持单继承，即一个类型只能继承一种类型。

面向对象的程序设计语言提供了应用对象的能力，类型的一个属性可以是对一个指定类型的对象的引用。

4. 与复杂类型有关的查询

与复杂类型有关的查询可以分为如下几类：路径表达式、以集合体为值的属性、嵌套与解除嵌套。

5. 函数与过程

对象关系数据库系统中允许用户定义函数与过程，它们既可以用某种数据操纵语言（如SQL）来定义，也可以通过外部的程序设计语言来定义，如Java、C或C++。有些数据库管理系统支持它们自己的过程语言，如Oracle中的PL/SQL和Microsoft SQL Server中的Transact-SQL，它们类似于SQL有关过程的部分，但在语法和语义上有所区别，详细信息可参见各自的系统手册。

6. 面向对象与对象关系

在持久化程序设计语言上的面向对象数据库和建立在关系模型之上的面向对象的对象关系数据库在数据库系统市场上都存在，数据库设计者要选择适合应用需求的系统。

程序设计语言的持久化扩展和对象关系系统有着不同的市场目标。SQL语言的声明性特征和有限的能力为防止程序设计错误对数据造成破坏提供了很好的保护，同时使得一些高级优化，如减少I/O，变得相对简单。对象关系系统的目标在于通过使用复杂数据类型来简化数据建模和查询，典型的应用有复杂数据的存储和查询等。

然而，对于某些类型的应用，如主要在内存中运行和对数据库进行大批量访问的应用来说，一个说明性语言（如SQL）会带来显著的性能损失。满足应用的高性能要求就是持久化程序设计语言的目标。持久化程序设计语言提供了对持久数据的低开销存取，并且取消了

数据转换要求。但是，持久化程序设计语言对由于程序错误而引起的数据破坏更为敏感，而且通常没有强大的查询能力。其典型应用包括 CAD 数据库。

10.2 分布式数据库系统

10.2.1 数据库系统体系结构

数据库系统是数据密集型应用的核心，其体系结构受数据库运行所在的计算机系统的影响很大，尤其受计算机体系结构中的联网、并行和分布的影响。从数据库管理系统的角度看，数据库系统体系结构一般采用外模式、模式和内模式的三级模式结构；从最终用户的角度看，数据库系统体系结构分为集中式、分布式、客户/服务器和并行结构。

1. 集中式数据库系统

集中式数据库系统是由一个处理器、与它相关联的数据存储设备以及其他外围设备组成，它被物理地定义到单个位置。系统提供数据处理能力，用户可以在同样的站点上操作，也可以在地理位置隔开的其他站点上通过远程终端来操作。系统及其数据管理被某个或中心站点集中控制。分时系统环境下的集中式数据库系统结构诞生于 20 世纪 60 年代中期。当时的硬件和操作系统决定了分时系统环境下的集中式数据库系统结构成为早期数据库技术的首选结构。在这种系统中，不但数据是集中的，而且数据的管理也是集中的。数据库系统的所有功能，从形式的用户接口到 DBMS 核心都集中在 DBMS 所在的计算机上。目前，大多数关系 DBMS 产品都是从这种系统结构发展起来的，这种系统现在仍然有人在使用。

2. 客户/服务器数据库体系结构

客户/服务器数据库体系结构由客户端和服务器端逻辑组件构成。客户端一般是个人电脑或工作站，而服务器端是大型工作站、小型计算机系统或大型计算机系统。站在最终用户的角度看，采用客户/服务器(Client/Server，C/S)体系结构后，可以使某些任务在服务器上执行，另一些任务在客户机上执行。采用客户/服务器结构后，数据库系统功能分为前端和后端。前端主要包括图形用户界面、表格生成和报表处理等工具；后端负责存取结构、查询计算和优化、并发控制以及故障恢复等。前端与后端通过 SQL 或应用程序来接口。客户/服务器架构是开放系统架构的一部分，在开放系统架构里，所有计算硬件、操作系统、网络协议和其他软件相互连接形成网络并共同工作实现用户目标。它非常适用于联机事务处理和决策支持应用，它往往产生很多相关的小事务并要求高度的并发性。采用客户/服务器体系结构的主要特点是客户机与服务器之间的职责明确，客户机主要负责数据表示服务，而服务器主要负责数据库服务。与集中式系统相比，客户/服务器数据库系统更为灵活。

开放式数据库连接(Open Database Connectivity，ODBC)和 Java 程序数据库连接(Java Database Connectivity，JDBC)标准定义了应用程序和数据库服务器通信的方法，也就是定义了应用程序接口。应用程序可以用其打开与数据库的连接、发送查询和更新以及获取返回结果等。

数据服务器系统中的客户与服务器之间具有高速的连接，客户机的处理能力与服务器

相当，并且要执行的任务是计算密集型的。在这样的环境中，所有的功能都要放在客户端，数据被传送到客户机，由客户机进行所有处理，再将结果传回到服务器端。由于客户机与服务器之间通信的时间代价与本地存储器引用的代价相比要高得多，所以在实际应用中需要注意这方面的问题。

3. 分布式数据库系统

分布式数据库系统是计算机网络技术与数据库技术互相渗透和有机结合的产物。分布式数据库系统在许多情形下与客户/服务器架构类似，它们都使用多个计算机系统以及用户能够访问远程系统的数据。然而，分布式数据库系统可以更好地实现数据共享，超过了客户/服务器系统可以达到的程度。分布式数据库系统包括物理上分布、逻辑上集中的分布式结构和物理上逻辑上都分布的分布式数据库结构两种。前者的指导思想是：把数据模式（称为全局数据模式）按数据来源和用途，合理地分布在系统的多个结点上，使大部分数据可以就地或就近存取。数据在物理上分布后，由系统统一管理，使用户感觉不到数据的分布。后者一般由两部分组成：一是本结点的数据模式，二是本结点共享的其他结点上的有关数据模式。结点间的数据共享由双方协商确定，这种数据库结构有利于数据库的集成、扩展和重新配置。

4. 并行数据库系统

并行体系结构的数据库系统由多个物理上连在一起的中央处理单元（CPU）和并行的数据存储设备组成。因此，它们提高了处理能力和输入/输出速度。而分布式系统的各个CPU在地理上是分开的。并行体系结构的数据库类型分为共享内存式多处理器和无共享式并行体系结构。并行数据库系统用于必须要对非常大的数据库进行查询的应用或者是每秒钟必须处理大量事务的应用。在并行数据库系统里，吞吐量和响应时间是非常高的。并行数据库系统的主要缺点在于，在并行数据库系统中，存在初始化单个进程相关的启动代价，而且启动时间可能掩盖实际的处理时间，反过来又会影响加速。而且，由于在并行系统中执行的进程经常访问共享资源，因此在新的进程与现有进程竞争共享资源时，就会产生干扰，使速度下降。

10.2.2 分布式数据库系统的概念和特点

1. 分布式数据库系统的概念

分布式数据库系统（Distributed Database System，DDBS）是针对地理上分散，而管理上又需要不同程度集中管理的需求而提出的一种数据管理信息系统。在明确给出分布式数据库的定义之前，先介绍一下一般的分布式数据库系统的组成，如图10-2所示。

可以看出，分布式数据库系统首先是由多个不同结点或场地的数据库系统通过网络连接而成的（如不加特别说明，本章中的场地和结点表示同一含义），每个结点都有各自的局部数据库管理系统（Local Database Management System，LDBMS），同时还有全局数据库管理系统（Global Database Management System，GDBMS）。图10-2中的局部用户是针对某一个结点而言的，局部用户只关心他所访问的结点上的数据，而全局用户则可能需要访问多个

结点上的数据。每个结点上的 LDBMS 响应局部用户的应用请求，GDBMS 则为全局用户提供服务，全局用户可以从任意一个结点访问分布式数据库系统中的数据。

从概念上讲，分布式数据库是由一组数据组成的，这组数据分布在计算机网络的不同计算机上，网络中的每个结点具有独立处理的能力（称为场地自治），可以执行局部应用。同时，每个结点也能通过网络通信子系统执行全局应用。

满足下列条件的数据库系统被称为完全分布式数据库系统。

（1）分布性，即数据存储在多个不同的结点上。

（2）逻辑相关性，即数据库系统内的数据在逻辑上具有相互关联的特性。

（3）场地透明性，即使用分布式数据库中的数据时无须指明数据所在的位置。

（4）场地自治性，即每一个单独的结点能够执行局部的应用请求。

分布式数据库系统的分布性可以让人们区分单一的集中式数据库与分布式数据库，而根据逻辑相关性，可以将分布式数据库与一组局部数据库或存储在计算机网络中不同结点的文件系统区分开来。场地的透明性和场地的自治性则可以和多机处理系统或并行系统区分开来。

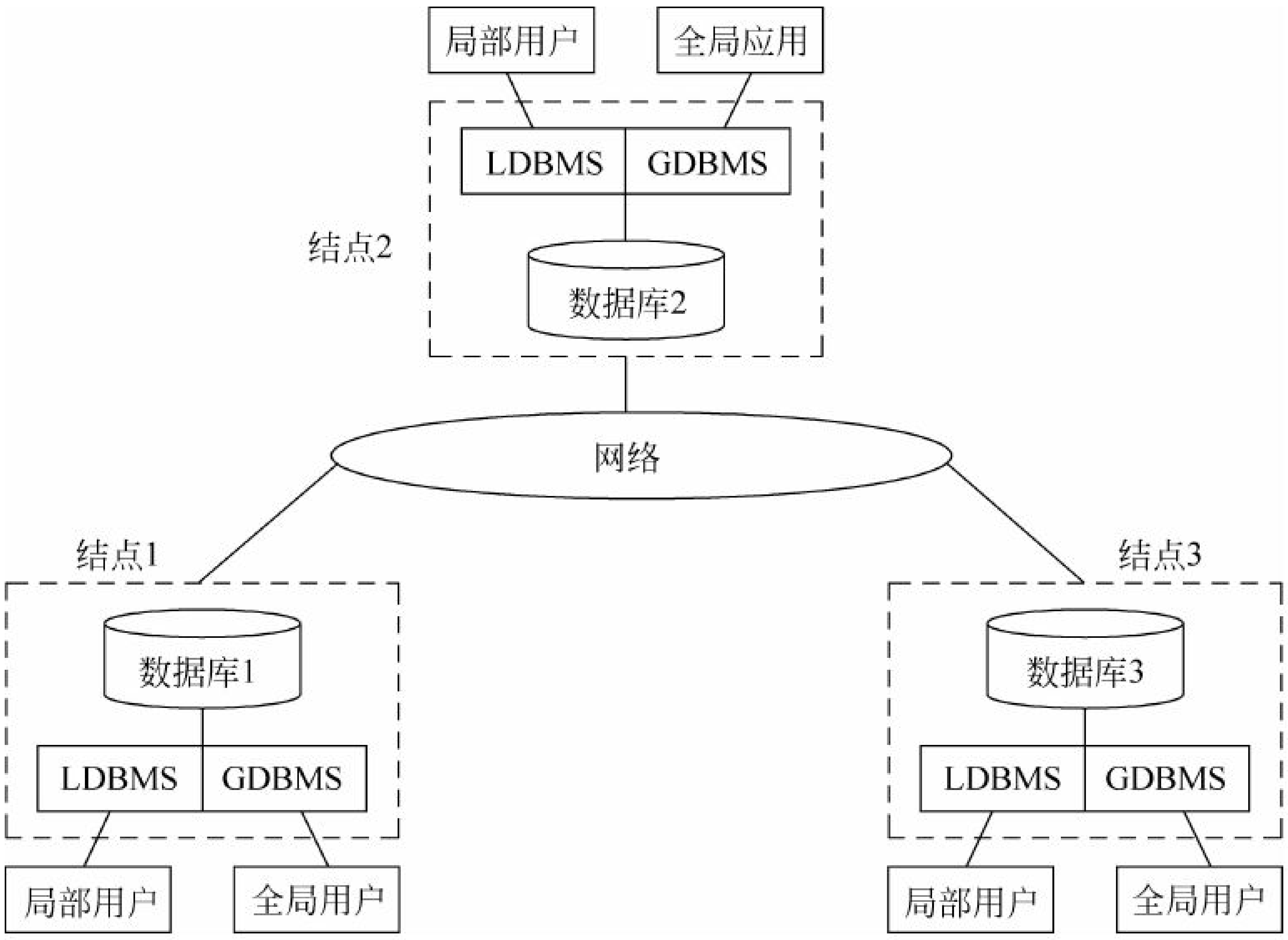

图 10-2 分布式数据库系统组成

2. 分布式数据库的特点

分布式数据库系统是传统集中式数据库系统的发展，因此它具有集中式数据库系统的特点。同时，它的分布性又使这些特点具有新的含义。传统的数据库系统针对文件系统的弱点，采用了集中控制以实现数据共享，这是其最主要的特色。对于分布式数据库系统来说，由于数据的分散性，分布式数据库系统具有分散与集中统一的特性。

（1）数据的集中控制性

能够对信息资源提供集中控制是使用数据库进行数据管理的最重要的功能之一。数据

库是随着信息系统的演变而发展起来的，在这些信息系统中，每个应用程序都有自己的专用文件，这样就不利于数据的管理和共享。由于数据本身已被当作企业的重要资源，在这样的需求推动下，传统的数据库系统应运而生。分布式数据库系统是在传统数据库系统的基础上的发展，所以，它也具有集中控制的特性。

在传统的数据库系统中，数据库管理员（DBA）的基本任务是保证数据的安全，并负责对数据进行管理，使用户和应用能够高效地访问数据。而在分布式数据库中，存在全局数据库管理员和局部数据库管理员。这是一种分层控制结构，一般来说，全局数据库管理员负责管理所有数据库，而局部数据库管理员只负责各自结点的局部数据库。但是在有些情况下，局部数据库管理员可以有更高的自主性，甚至能够完成结点间的协调工作，从而不再需要全局数据库管理员。

（2）数据独立性

数据独立性也是集中式数据库相对于文件系统的一大特征。独立性指的是数据的组成对应用程序来说是透明的。应用程序只需要考虑数据的逻辑结构，而不用考虑数据的物理存放，因而数据在物理组织上的改变不会影响应用程序。

在分布式数据库系统中，数据的独立性同样具有重要意义。分布式数据库的数据独立性除了具有传统意义上数据独立性的含义之外，还有分布式透明的含义。所谓分布式透明是指：虽然应用程序面对的是分散存放的数据，但就像使用集中式数据库一样，不必考虑数据库的分布特性。

（3）数据冗余可控性

将数据组织在数据库中可以方便地实现数据共享，因此要尽量减少数据冗余。这不仅能够降低存储代价，还可以提高查询效率，便于维护数据的一致性，这是数据库系统优于文件系统的特点之一。但对数据库系统来说，也不可能做到绝对无冗余数据。

对于分布式数据库来说，由于数据存储的分散性，需要在网络上传输数据。与集中式数据库相比，查询中就增加了传输代价。因此，分布式数据库中的数据一般存储在经常使用的场地上，但应用对两个或两个以上场地的同一数据有存取要求也是时常发生的，而且当传输代价高于存储代价时，可以将同一数据存储在两个（甚至更多）场地上，以节省传输的开销。另外，数据有多个副本，也可以提高系统的可用性，即当系统中某个结点发生故障时，因为数据有其他副本在非故障场地上，数据仍然是可用的，从而保证了数据的完备性。由于这种冗余度是在系统控制下的，因此给系统造成的不利影响是可控制的。

另外，由于可用副本的存在，也相应地提高了场地自治性的性能。

（4）场地自治性

在分布式数据库系统中，多个场地的局部数据库逻辑上为一个整体，这个整体称为全局数据库，并可以为分布式数据库系统的所有用户使用，这种应用称为分布式数据库的全局应用，其用户为全局用户。同时，分布式数据库系统还允许用户只使用本地的局部数据库，这种应用为局部应用，其用户为局部用户。局部用户所使用的数据甚至可以不参与到全局数据库中去，这种局部应用独立于全局应用的特性就是局部数据库的自治性。

由于具有自治性，对每个场地来说就有两种数据，一种是参与全局数据库的局部数据，另一种则是不参与全局数据库的数据。

(5) 存取的有效性

在传统的数据库系统中,采用二次索引和文件链接等复杂的存储结构是提高存取效率的主要方法。但在分布式数据库系统中,仅仅采用复杂的存取结构并不是正确的方法。分布式数据库系统的全局查询可以分解成等效的子查询,即全局查询的执行计划可分解成多个子查询执行计划(它是根据系统的全局优化策略产生的),而子查询计划又是在各场地上分布执行的。因此,分布式数据库系统中的查询优化有两个级别:全局优化和局部优化。

全局优化主要决定在多个副本中选取合适的场地副本,使得场地间的数据传输量传输次数最少,从而使系统通信开销减少。而局部优化就和传统的集中式数据库中的优化是一致的。

10.2.3 分布式数据库的体系结构

分布式数据库的模式结构是目前国内外尚在讨论的问题,还没有形成统一的标准,这是因为对分布式数据库系统数据独立性要求的程度不同,其抽象层次也不同。我国在多年研究与开发分布式数据库中,制定了《分布式数据库系统标准》,提出把分布式数据库抽象为4层的模式结构,如图10-3所示。这一模式结构得到了国内外专家一定程度的支持和认同。

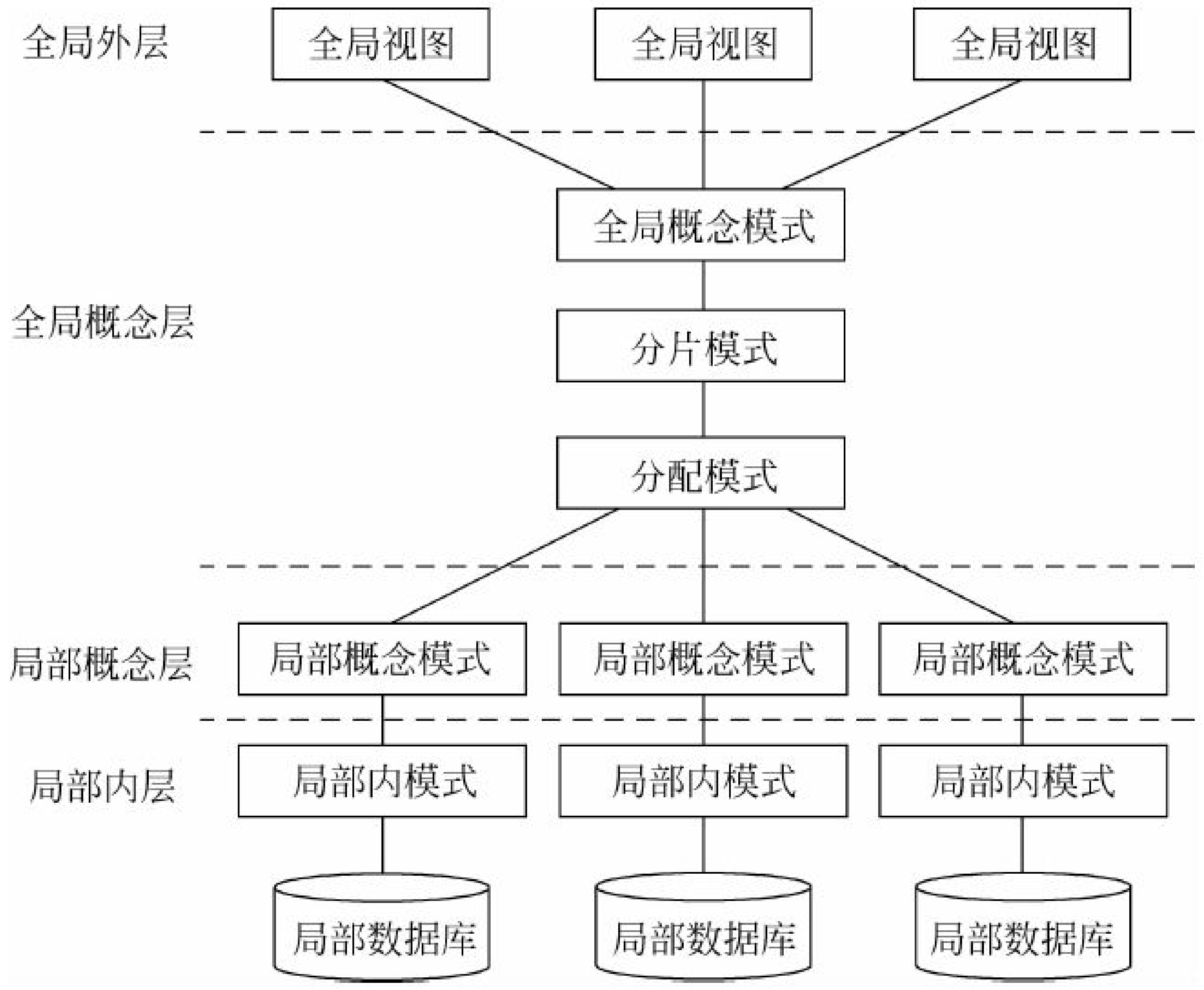

图10-3 分布式数据库结构模式

4层模式是指将数据库系统划分为全局外层、全局概念层、局部概念层和局部内层。在各层间还有相应的层间映射。4层模式的划分不仅适用于完全透明的分布式数据库系统,而且也适合各种透明性要求的分布式数据库系统。无论是对同构型分布式数据库系统,还是异构型分布式数据库系统都能适用。

10.2.4 分布式数据库系统的分类

对于分布式数据库系统,从不同角度观察这些系统,有不同的分类方法。例如,可以按

照许多方式进行分类，如可以根据全局控制方式分类，也可以按数据在逻辑上是集中/分布来分类，通常根据局部场地的DBMS及数据模型来对分布式数据库系统进行分类，可以将分布式数据库系统分为两类：同构分布式数据库管理系统和异构分布式数据库管理系统。

1. 同构分布式数据库管理系统

同构分布式数据库管理系统是指系统中每个结点上的局部数据库管理系统(LDBMS)类型都相同，即它们支持相同的数据类型、访问方法、优化策略、并发控制算法，以及相同的命令语言和查询语言等。通常同构型各结点的数据模型和全局数据模型是相同的。同构分布式数据库管理系统类似于一个集中式数据库，只不过同构分布式数据库将数据存放分布在网络中不同的结点内，而不是存放在一个结点内。同构分布式数据库系统可以根据是否自治划分为自治式分布式数据库系统和非自治式分布式数据库系统。

2. 异构分布式数据库管理系统

异构分布式数据库系统是指系统中每个结点上可以有不同类型的LDBM，它的特点是在各结点上运行着不同的数据库管理系统。它们可以是层次、网状和关系型的数据库管理系统，也可以是同一数据模型但不同厂商提供的DBMS产品。因此，在异构分布式数据库系统中，应该提出对不同数据模型、访问方法、优化策略、并发控制算法以及查询语言的支持。在异构分布式数据库系统中，不同结点上的数据库系统具有独立性、自治性和分布透明性等特点，用户对任何数据库的操作不必关心其数据模型、物理位置等细节，如同在本地执行一样。因此，它屏蔽了各种数据库在物理上和逻辑上的差异，使用户用自己所熟悉的一种数据操作语言就能够操纵任何一种数据库。

异构分布式数据库管理系统在数据库设计和实现比同构型系统复杂。它要解决不同的DBMS之间的数据模型、事务管理协议、查询语言等的转换问题。这种转换通常需要寻找某种合适的公共数据模型，常选用一种称做规范的中间数据模型和中间语言。然后将不同类型的DBMS数据模型和查询语言先转换成中间规范的形式，再完成它们之间的转换。通常不建议采用各结点之间一对一的转换方式，以减少转换次数。

10.3 Web与数据库

随着网络的高速发展和网络服务的日趋完善，网络上信息量呈几何级数增长。为了有效地组织、存储、管理和使用网上的信息，数据库技术被普遍地应用于网络领域。人们在Internet上建立了数以万计的网站，尤其是大中型网站的后台都有数据库系统的支持。数据库系统可以将网站的各种数据很好地组织起来，并自动生成Web页面。

10.3.1 Web数据库

数据库技术是计算机处理与存储数据最有效、最成功的技术，而计算机网络的特点是资源共享，因此数据与资源共享这两种技术的结合即成为今天广泛应用的Web数据库(也叫网络数据库)。

Web 的产生与互联网的发展是密切相关的。互联网是由全球众多的计算机局域网互相连接组成的一个超大规模的网络系统。在这个系统中运行着多种应用系统，如上网使用的网页浏览系统 WWW、上传与下载用的文件传输系统 FTP，以及收发电子邮件所使用的电子邮件系统 E-mail 等。互联网中运行的每一种应用系统都是由互联网中相应的服务器系统和客户机系统构成的，也就是说，从物理连接来看，互联网是由众多的计算机组成的，而从逻辑上看，互联网是由多个功能子网组成的。

在浏览网页时，服务器上的 WWW 服务允许单击"超级链接"，每单击一项，WWW 程序就会执行所要求的任务，一直到需要得到满足为止。在这一过程中，涉及两个不同的程序。一个程序安装在客户机上，处理鼠标单击事件，发出 http 请求。接到响应后，立即显示链接的网页内容，这个程序叫做 WWW 客户机程序，如上网使用的浏览器(IE 或 Netscape)。另一个程序在服务器上，如 Internet 信息服务(Internet Information Server，IIS)或阿帕奇(Apache)Web 服务器软件，它对 WWW 客户机的请求进行处理并返回处理结果，即在接到 http 请求后做出响应。

Web 数据库就是利用浏览器作为用户输入接口，输入所需要的数据，浏览器将这些数据传送给网站。网站再对这些数据进行处理，例如将数据存入后台数据库或者对后台数据库进行查询操作等。最后，网站将操作结果回传给浏览器，通过浏览器将结果告知用户。网站上的后台数据库就是 Web 数据库。

总之，数据库技术用于对大量的数据进行组织和管理；Web 技术具有较佳的信息发布途径，这两种技术的天然互补性决定了相互融合是其发展的必然趋势。将 Web 技术与数据库技术融合在一起，使数据库系统成为 Web 的重要有机组成部分，不仅可以把二者的所有优点集中在一起，而且能够充分利用大量已有的数据库信息资源，使用户在 Web 浏览器上方便地检索和浏览数据库的内容，这对许多软件开发者来说具有极大的吸引力。因此，将 Web 技术与数据库相结合，开发动态的 Web 数据库应用已成为当今 Web 技术研究的热点。数据库厂家也推出了各自的产品和中间件以支持 Web 技术和 DBMS 的融合，使两者能够互相取长补短，发挥各自的优势。Web 数据库技术是 Web 技术与数据库技术相互结合的产物，它的出现将信息的存储、管理和检索提升到了一个全新的高度。在目前 Web 数据库应用的开发领域中，有多种用于开发 Web 数据库系统的技术，如 CGI、PHP、ASP、ASP.NET、Java EE 等。

10.3.2　Web 数据库与传统数据库比较

Web 数据库技术是 Web 技术与数据库技术相互结合的产物，是人们对信息的需求日益高涨的必然结果。Web 数据库与传统的关系数据库相比，具有以下的优点。

(1) 界面统一。Web 数据库借用现成的浏览器软件，无须再开发数据库前端。如果能够通过 WWW 来访问数据库，我们就不需要开发客户端程序，使用的数据库应用都可以通过浏览器来实现。这样一来，不但客户端界面统一，同时也减少了培训的时间和费用，能使广大用户很方便地访问数据库信息。

(2) 标准统一。Web 数据库开发过程简单，标准相对统一。HTML 是一种国际标准，Internet 中所有的 Web 服务器与客户端浏览器都支持这一标准，开发者甚至只需学习 HTML 一种语言，使用者只需通过浏览器界面就可以方便地检索和浏览数据库的内容。

(3) 交叉平台支持。几乎在各种操作系统上都有现成的浏览器可供使用,为一个 Web 服务器书写的 HTML 文档,可以被几乎所有平台的浏览器所浏览,实现了跨平台操作。

因此,Web 数据库应用的普遍化是当今数据库应用发展的一个必然趋势。

10.3.3 Web 服务器脚本程序与服务器的接口

Web 页面与数据库的连接是 Web 数据库的基本要求。目前,基于 Web 数据库的连接方案主要有两种类型:服务器端和客户端方案。服务器端方案的实现技术有 CGI、PHP、ASP、ASP.NET、Java EE 等。客户端方案的实现技术有 JDBC 等。

通常 Web 数据库的环境由硬件元素和软件元素组成。硬件元素包括 Web 服务器、客户机、数据库服务器和网络;软件元素包括客户端和服务器端两个方面。客户端必须有能够解释执行 HTML 代码的浏览器(如 IE 和 Netscape 等)。在 Web 服务器中,必须具有能自动生成 HTML 代码程序的功能,如 ASP 和 CGI 等,具有能自动完成数据操作指令的数据库系统,如 Access 和 SQL Server 等。

Web 服务器使用 HTTP 协议对客户机的请求给予回答。每个 Web 服务器在 Internet 上都有一个唯一的地址,这个地址可以是一个域名,也可以是 4 个以点分隔的 0～255 之间的数字,例如 www.yahoo.com 或 202.106.168.67。如果客户机提出一个合法的请求,那么 Web 服务器就会把请求的内容传送给客户机。Web 服务器不仅能够传送各种文件,还可以传送某个程序的输出结果,这为 Web 和数据库的结合创造了条件。Web 服务器的种类很多,比较著名的有 IIS 和 Apache 等。

典型的 Web 和数据库的运行模式如图 10-4 所示。

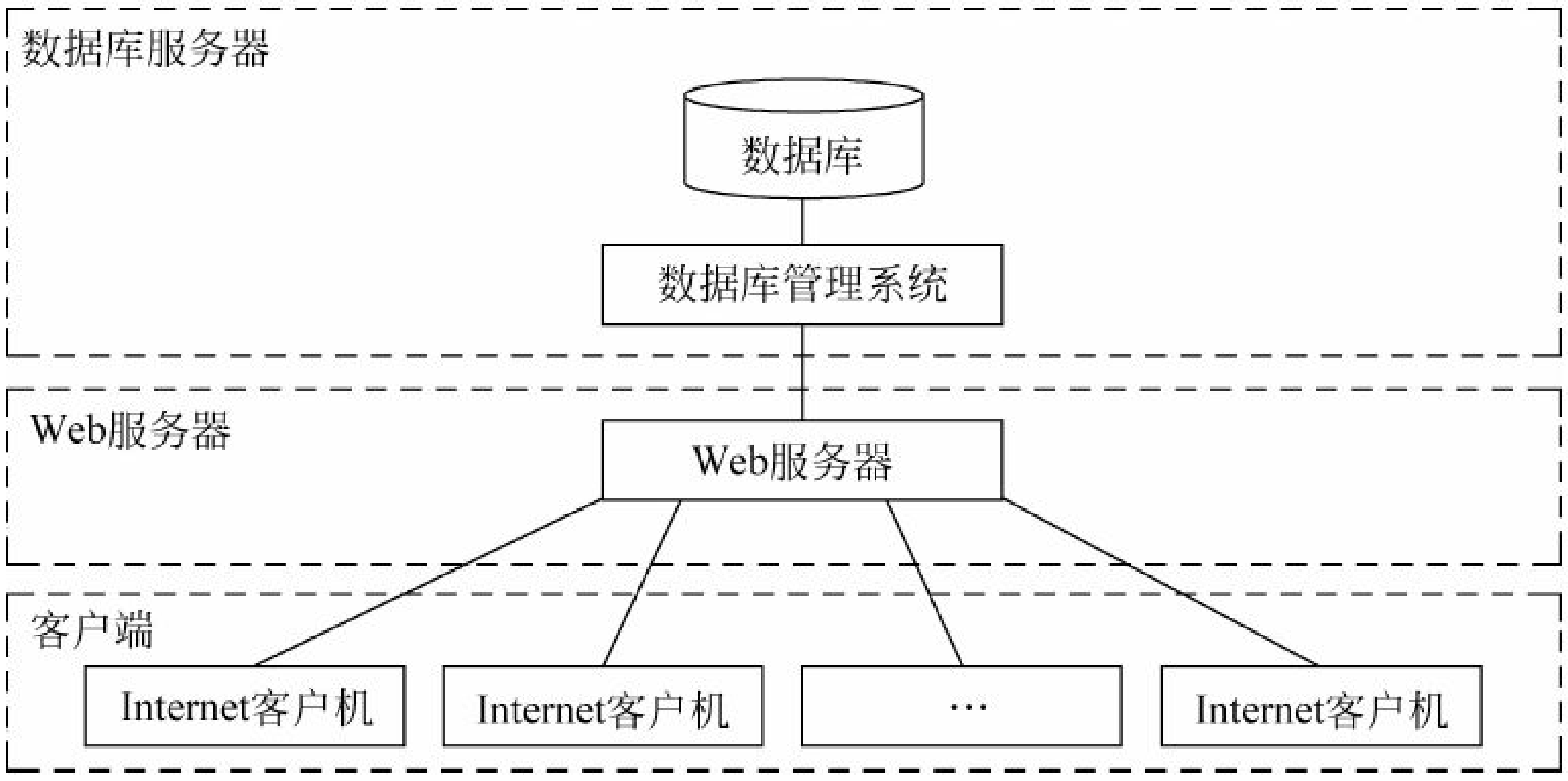

图 10-4 Web 和数据库运行模式示意图

在脚本程序中一般都需要采用相应的接口连接数据库。连接数据库的常用方法有 ODBC、DAO、RDO 及 ADO 等。

1. ODBC

ODBC(Open Database Connectivity,开放式数据库连接)是微软开发的一套统一的程序接口。它建立了一组规范,并提供了一组对数据库访问的标准 API(应用程序编程接口)。这些 API 利用 SQL 来完成其大部分任务。ODBC 本身也提供了对 SQL 语言的支持,用户

可以直接将 SQL 语句送给 ODBC。经过多年的改进，它已成为存取数据库服务器的标准。事实上，ODBC 技术已成为后来的 DAO、RDO 及 ADO 等数据库访问技术的基础。

2．DAO

DAO(Data Access Objects，数据访问对象)是微软公司开发的一种应用程序编程接口(API)，是微软的第一个面向对象的数据库接口。利用它可以访问数据库的标准对象，如 Access、VB、Excel 和 Word 等。DAO 最适用于单系统应用程序或小范围本地分布使用。

3．RDO

RDO(Remote Data Objects，远程数据对象)是微软公司为增强 DAO 的功能而推出的新产品。该产品强化了 SQL Server 的访问功能，提高了 SQL Server 的执行效率。RDO 已被证明是许多 SQL Server、Oracle 以及其他大型关系数据库开发者经常选用的接口。RDO 提供了用来访问存储过程和复杂结果集的更多和更复杂的对象、属性，以及方法。

4．ADO

ADO(ActiveX Data Objects，ActiveX 数据对象)是微软在 Internet 领域采取的新举措。它本身并不是一项新技术，从对象结构的角度来看，它比 DAO 提供的对象更少；从存取 SQL 服务器的角度来看，它提供的功能也不如 RDO。但它汲取了 DAO 和 RDO 精华的部分，成为一个更适合于 Internet 的小而精的对象群。因此，ADO 实际上是脚本程序连接数据库的一种最佳选择。在新的编程框架.NET Framework 中，微软也提供了一个面向 Internet 的版本的 ADO，称为 ADO.NET，其对象模型和传统 ADO 差别很大。

10.3.4　应用开发平台

由于 Web 应用开发的独特性，应用开发平台成为众多厂商关注的焦点。目前市场上存在很多 Web 应用标准和集成开发环境，下面介绍目前比较流行的三种。

1．ASP.NET

ASP(ActiveX Server Pages)是由微软创建的 Web 应用开发标准，是一种使嵌入网页中的脚本可由因特网服务器执行的服务器端脚本技术。它是 ASP.NET 的前身。ASP.NET 是建立在微软新一代.NET 平台构上，利用普通语言运行时(CLR)在服务器后端为用户提供建立强大的企业级 Web 应用服务的编程框架。ASP.NET 是一个已编译的、基于.NET 的环境，把基于通用语言的程序在服务器上运行。将程序在服务器端首次运行时进行编译，比 ASP 即时解释程序的速度要快很多。而且是可以用任何与 .NET 兼容的语言(包括 VB.NET、C# 和 JScript .NET)创作应用程序。ASP.NET 可完全利用.NET 架构的强大、安全和高效的平台特性，运行时早绑定、即时编译、本地优化、缓存服务、零安装配置和基于运行时代码受管与验证的安全机制等都为 ASP.NET 带来了卓越的性能。

2．PHP

PHP 是一种 HTML 内嵌式的语言，是一种在服务器端执行的嵌入 HTML 文档的脚

本语言。PHP由于具有良好的性能及开源的特点，成为目前互联网中应用非常流行的一种应用开发平台。PHP具有非常强大的功能，所有的CGI的功能PHP都能实现，而且支持几乎所有流行的数据库以及操作系统。它具有很多方面的优点：首先和其他技术相比，PHP本身免费，是开放的源代码，其次PHP简单易学，程序开发快，运行快，技术本身容易学习。因为PHP可以嵌入HTML语言，它相对于其他语言，编辑简单，实用性强，更适合初学者。此外，PHP还可以跨平台互用，是具有良好数据库交换能力的开发语言，能够与Apache及其扩展库紧密结合，具有良好的安全性；PHP消耗相当少的系统资源，效率高。PHP不足之处在于安装配置复杂，缺少企业级的支持，作为自由软件，缺乏正规的商业支持，无法实现商品化的商业开发。

3. Java EE

Java EE(Java Platform，Enterprise Edition)是Sun公司推出的企业级应用程序版本。这个版本以前称为J2EE。能够帮助我们开发和部署可移植、健壮、可伸缩且安全的服务器端Java应用程序。简单地说，Java EE就是一套API的规范，一个分布式计算体系，以及用于分布式部署的组件包的定义。它是标准化组件、容器、服务的集合，用于在一个明确的分布式计算系统中创建和部署分布式应用。Java EE平台由一整套服务(Service)，应用程序接口(API)和相关协议构成。它对开发基于Web的多层、分布式应用提供了强大的功能支持。通常，Java EE的目标是大规模的业务系统。所以这个级别的软件不会运行在单机上。因此需要把一个软件分割成许多功能模块，并部署到相应的计算机硬件平台上。这也就是分布式计算的本质。Java EE还包含了一套实现软件部署的标准化组件，定义了各种软件模块如何交互的标准接口，以及不同软件模块通信的标准服务。

10.4 数据仓库

信息技术的广泛应用将企业带入了信息爆炸的时代，管理者面对着大量等待处理的信息。对这些信息的处理分为事务型处理和信息型处理两大类。事务型处理就是通常所说的业务操作处理，是对管理信息进行的日常操作。对信息进行查询和修改，目的是为了满足组织特定的日常管理需要。信息型处理则是对信息做进一步的分析，为管理人员决策提供支持。这类信息的处理在现代企业中应用越来越广泛，越来越引起管理人员的重视。管理信息的信息型处理必须访问大量的历史数据才能完成，不像事务型处理那样，只对当前信息感兴趣。

10.4.1 数据仓库概述

1. 数据仓库的定义

数据仓库(Data Warehouse，DW)的概念是20世纪90年代初被提出来的。传统数据库在联机事务处理(Online Transaction Processing，OLTP)中获得了较大的成功，但对管理人员的决策分析要求却无法满足。因为管理人员希望对组织中的大量数据进行分析，了解组织业务的发展趋势。而传统的数据库中只能保留当前的管理信息，缺乏决策分析所需要的大量历史信息。为了满足管理人员决策分析的需要，在数据库基础上产生了能满足决策分

析需要的数据环境——数据仓库。

数据仓库是决策支持系统和联机分析应用数据源的结构化数据环境。数据仓库研究和解决从数据库中获取信息的问题。数据仓库之父 William H. Inmon 于 1991 年出版的 *Building the Data Warehouse* 一书对数据仓库给出了如下定义：数据仓库是一个面向主题的、集成的、相对稳定的、反映历史变化的数据集合，用于支持经营管理中的决策制定过程。

2. 数据仓库与数据库

数据仓库虽然是从数据库发展而来的，但是二者在许多方面有很大的差异，如表 10-1 所示。从表中可以看出，数据库是面向事务的设计，数据仓库是面向主题设计的。数据库一般存储数据的当前值，而数据仓库存储的一般是历史数据。数据库中的数据是动态变化的，而数据仓库中的数据是静态的，不能直接更新。数据库中数据结构是高度结构化、复杂、适合操作计算的，而数据仓库中的数据结构相对比较简单，适合分析。另外，数据库设计时尽量避免冗余，一般采用符合范式的规则来设计，而数据仓库在设计时有意引入冗余，采用反范式的方式来设计。数据库在基本容量上要比数据仓库小得多。数据库是为了高效的事务处理而设计的，服务对象为企业业务处理方面的工作人员，而数据仓库是为了分析数据进行决策而设计的，服务对象为企业高层决策人员。

表 10-1 数据仓库与数据库的比较

内 容	数据仓库	数据库
数据内容	历史的、存档的、归纳的、计算的	当前值
数据目标	面向主题域、分析应用	面向业务操作人员，重复处理
数据特性	静态、不能直接更新，只能定时添加、刷新	动态变化，按字段更新
数据结构	简单、适合分析	高度结构化、复杂、适合操作计算
使用频率	中、低	高
数据访问量	有的事务可能需要访问大量记录	每个事务只访问少量的记录
对响应时间的要求	以秒、分钟、甚至小时为计算单位	以秒为单位计算

10.4.2 数据仓库的基本特性

数据仓库具有以下四个基本的特性：数据是面向主题的、数据是集成的、数据是相对稳定的、数据是反映历史变化的。

1. 数据是面向主题的

数据仓库中的数据是按面向主题进行组织的。从信息管理的角度来看，主题就是一个较高的管理层次上对信息系统中的数据按照某一具体的管理对象进行综合、归类所形成的分析对象。从数据组织的角度来看，主题就是一些数据集合，这些数据集合对分析对象进行了比较完整的、一致的数据描述，这种数据描述不仅涉及数据自身，还涉及数据间的联系。例如，企业中的客户、产品和供应商等都可以作为主题来看待。

数据仓库的创建和使用都是围绕主题实现的，因此，必须了解如何按照决策分析来抽取主题，所抽取的主题应该包含哪些数据内容，这些数据应该如何组织。在进行主题抽取时，

必须按照决策分析对象进行。例如，企业销售管理人员所关心的是本企业哪些产品销售量大、利润高？哪些客户采购的产品数量多？竞争对手的哪些产品对本企业的产品构成威胁？根据这些管理决策分析对象，就可以抽取“产品”和“客户”等主题。

2. 数据是集成的

数据仓库的集成性是指根据决策分析的要求，对分散于各处的源数据进行抽取、筛选、清理及综合等集成工作，使数据仓库中的数据具有集成性。

数据仓库所需要的数据不像业务处理系统那样直接从业务发生地获取数据。例如，联机事务处理系统 OLTP、企业业务流程重组 BPR 以及基于因特网的电子商务中的数据是与业务处理联系在一起的，只为业务的日常处理服务，而不是为决策分析服务。这样，数据仓库在从业务处理系统那里获取数据时，并不能将原数据库中的数据直接加载到数据仓库中，而要进行一系列的数据预处理。也就是说，从原数据库中挑选出数据仓库所需要的数据，然后将来自不同数据库中的数据按某一标准进行统一，如将数据源中数据的单位、字长与内容统一起来，将源数据中字段的同名异义、异义同名现象消除，然后将源数据加载到数据仓库，并对数据仓库中的数据进行某种程度的综合，进行概括和聚集处理。

3. 数据是相对稳定的

数据仓库的数据主要供决策分析用，所涉及的数据操作主要是数据查询，一般情况下并不进行修改操作。数据仓库的数据反映的是一段相当长的时间内历史数据的内容，是不同时间的数据库快照的集合，以及基于这些快照进行统计、综合和重组的导出数据，而不是联机处理的数据。数据库中进行联机处理的数据经过集成输入到数据仓库中。因为数据仓库只进行数据查询操作，DBMS 中的完整性保护、并发控制，在数据仓库管理中都可以省去。但是，由于数据仓库的查询数据量往往很大，所以对数据查询提出了更高的要求，需要采用复杂的索引技术。

4. 数据是反映历史变化的

数据仓库中数据的相对稳定是针对应用而言的，数据仓库的用户进行分析处理时是不进行数据更新操作的，但并不表明在从数据集成输入数据仓库开始到最终被删除的整个数据生存周期中，数据仓库中所有的数据是永远不变的。数据仓库中的数据是反映历史变化的，这主要表现在如下 3 个方面：

(1) 数据仓库随时间变化不断增加新的数据内容。数据仓库系统必须不断捕捉 OLTP 数据库中变化的数据，追加到数据仓库中去。

(2) 数据仓库随时间变化不断删除旧的数据内容。

(3) 数据仓库中包含大量的综合数据，这些数据有很多信息与时间有关，如数据经常按时间段进行综合，或隔一定的时间进行抽样等，这些数据要随时间不断地进行重新综合。

10.4.3 数据仓库的体系结构

数据仓库通常采用 3 层体系结构，底层为数据仓库服务器，中间层为 OLAP 服务器，顶层为前端工具。底层的数据仓库服务器一般是一个关系数据库系统，数据仓库服务器从操

作型数据库或外部数据源提取数据，对数据进行清理、转换和集成等，然后装入数据仓库中。中间层的 OLAP 服务器的实现可以是关系型 OLAP，即扩充的关系型 DBMS，提供对多维数据的支持，也可以是多维的 OLAP 服务器，它是一种特殊的服务器，直接支持多维数据的存储和操作。顶层的前端工具主要包括各种查询工具和报表工具、数据分析工具及数据挖掘工具等。其中数据分析工具主要针对 OLAP 服务器，报表工具、数据挖掘工具主要针对数据仓库。

从结构的角度看，可有 3 种数据仓库模型：企业仓库、数据集市和虚拟仓库。

企业仓库收集跨越整个企业各个主题的所有信息。它提供全企业范围的数据集成，数据通常都来自多个操作型数据库和外部信息提供者，并且是跨越多个功能范围的。它通常包含详细数据和汇总数据。企业数据仓库可以在传统的大型机上实现，如 UNIX 超级服务器或并行结构平台。它需要广泛的业务建模，可能需要多年的时间来设计和建造。

数据集市包含对特定用户有用的、企业范围数据的一个子集。它的范围限于选定的主题。例如一个商场的数据集市可能限定它的主题为顾客、商品和销售。包括在数据集市中的数据通常是汇总的。通常，数据集市可以在低价格的部门服务器上实现，基于 UNIX 或 Windows NT/2000/XP。实现数据集市的周期一般是数周，而不是数月或数年。但是，如果它的规划不是企业范围的，从长远角度讲，可能会涉及很复杂的集成问题。根据数据的来源不同，数据集市分为独立的和依赖的两类。在独立的数据集市中，数据来自一个或多个操作型数据库或外部信息提供者，或者是一个特定部门或本地产生的数据。在依赖数据集市中，数据直接来自企业数据仓库。

虚拟仓库是操作型数据库上视图的集合。为了有效地处理查询，只有一些可能的汇总视图被物化。虚拟仓库易于建立，但需要操作型数据库服务器具有剩余能力。

10.4.4 数据仓库设计

数据仓库的开发过程一般包括数据仓库的分析、数据仓库的设计、数据仓库的实施和数据仓库的应用、支持和增强等四个阶段。进行数据仓库的设计时，采用与传统数据库一样的设计方式。构建数据仓库通常进行三个层次的数据模型设计：概念模型、逻辑模型和物理模型。

数据仓库概念模型的设计一般包括界定系统边界；确定数据分析的主题；确定量度和数据粒度；确定数据分析维度；定义类别；创建信息包图。概念模型的设计是在较高的抽象层次上的设计，因此建立概念模型时不用考虑具体技术条件的限制。

根据数据仓库的概念模型并不能直接建立数据仓库的物理模型，它们之间还存在一个中间层的逻辑模型。由逻辑模型来指导数据仓库的物理实施。设计数据仓库的逻辑模型的主要任务是对前期收集的信息的细化，将信息包图转换成数据仓库的模型图。

数据仓库逻辑模型的设计主要包括粒度层次划分，数据分割策略的确定，关系模式的定义，数据源及数据抽取模型的确定等。数据仓库逻辑设计中要解决的一个重要问题是决定数据仓库的粒度划分层次，粒度层次划分适当与否直接影响到数据仓库中的数据量和所适合的查询类型。在确定数据分割策略时，要选择适当的数据分割的标准。一般要考虑以下几方面的因素：数据量、数据分析处理的实际情况、简单易行以及粒度划分策略等。数据量的大小是决定是否进行数据分割和如何分割的主要因素；数据分析处理的要求是选择数据

分割标准的一个主要依据，因为数据分割是跟数据分析处理的对象紧密联系的；此外还要考虑到所选择的数据分割标准应是自然的、易于实施的；同时也要考虑数据分割的标准与粒度划分层次是适应的。

物理模型根据逻辑模型创建，它是通过指定主码和指定模型的其他物理特性来扩展逻辑模型而得到的。物理模型设计阶段的主要任务是确定数据仓库的存储结构、数据的存储位置和索引策略。

10.4.5 数据挖掘

随着数据库技术的不断发展及数据库管理系统的广泛应用，数据库中存储的数据量急剧增大，在大量的数据背后隐藏着许多重要的信息。如果能把这些信息从数据库中抽取出来，将会为公司创造很多潜在的利润。而这种从海量数据库中挖掘信息的技术，就称之为数据挖掘(Data Mining，DM)。事实上，从技术角度看，数据挖掘可以定义为从大量的、不完全的、有噪声的、模糊的以及随机的实际数据中提取隐含在其中的、人们不知道的、但又潜在有用的信息和知识的过程。

1. 数据挖掘的分类

数据挖掘工具能够对将来的趋势和行为进行预测，从而很好地支持人们的决策。比如，经过对公司整个数据库系统的分析，数据挖掘工具可以回答诸如"哪个客户对我们公司的邮件推销活动最有可能做出反应，为什么"等类似的问题。有些数据挖掘工具还能够解决一些很消耗人工时间的传统问题，因为它们能够快速地浏览整个数据库，找出一些专家们不易察觉的极有用的信息。

对数据挖掘技术进行分类可以有多种角度，按照所挖掘的数据库种类可分为关系型数据库的数据挖掘、数据仓库的数据挖掘、面向对象数据库的挖掘、空间数据库的挖掘、正文数据库的数据挖掘和多媒体数据库的数据挖掘等；按所发现的知识类别可分为关联规则、特征描述、分类分析、聚类分析以及趋势和偏差分析等；按所发现的知识抽象层次可分为一般化知识、初级知识和多层次知识等。

数据挖掘技术是人们长期对数据库技术进行研究和开发的结果。起初各种商业数据是存储在计算机的数据库中的，然后发展到可对数据库进行查询和访问，进而发展到对数据库的即时遍历。数据挖掘使数据库技术进入了一个更高级的阶段，它不仅能对过去的数据进行查询和遍历，并且能够找出过去数据之间的潜在联系，从而促进信息的传递。随着海量数据搜集、强大的多处理器计算机和数据挖掘算法这三类技术的发展和成熟，数据挖掘技术已在数据仓库系统中得到了广泛的应用。数据挖掘的发展受到数据库系统、统计学、机器学习、可视化技术、信息技术及其他学科的影响，如神经网络、粗糙集理论、知识表示、归纳技术与高性能计算等。

2. 几种常用的数据挖掘算法

(1) 人工神经网络

人工神经网络是20世纪80年代后期迅速发展起来的人工智能技术，仿照生理神经网络结构的非线形预测模型，通过学习进行模式识别。它对噪声数据具有很高的承受能力，对

未经训练的数据具有分类模拟的能力，因此在网站信息、生物信息和基因以及文本的数据挖掘等领域得到了越来越广泛的应用。

（2）决策树

决策树归纳是一种经典的分类算法。它采用自顶向下、递归的、各个击破的方式构造决策树。树的每一个结点上使用信息增益度量选择属性，可以从所生成的决策树中提取出分类规则。

（3）SVM 分类方法

SVM 即支持向量机法，该方法是建立在统计学习理论基础上的机器学习方法。通过学习，SVM 可以自动寻找出那些对分类有较好区分能力的支持向量，由此构造出的分类器可以最大化类与类的间隔，因而有较好的适应能力和较高的分准率。该方法只需要由各类域的边界样本的类别来决定最后的分类结果。SVM 法对小样本情况下的自动分类有着较好的分类结果。

（4）遗传算法

遗传算法是基于自然进化的基本理论，并采用遗传结合、遗传变异以及自然选择等设计方法的优化技术。遗传算法很容易实现并行运算，也可以用于分类等优化问题的求解。在数据挖掘中，它也可用于对其他算法的适应度进行评估。

（5）KNN 分类

KNN 分类即 K 最近邻法。该方法的思路非常简单直观：如果一个样本在特征空间中的 K 个最相似（即特征空间中最邻近）样本中的大多数属于某一个类别，则该样本也属于这个类别。该方法在分类决策上只依据最邻近的一个或者几个样本的类别来决定待分类样本所属的类别。该算法较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

3. 数据挖掘与数据仓库的关系

根据数据挖掘的定义可以看出，数据挖掘包含一系列旨在数据库中发现有用而未发现的模式的技术，如果将其与数据仓库紧密联系在一起，将会获取意外的成功。若将数据仓库比作矿坑，那么数据挖掘就是深入矿坑采矿的工作。毕竟如果没有足够丰富完整的数据，是很难期待数据挖掘能挖掘出什么有意义的信息的。成功的数据挖掘的关键之一在于只有访问正确、完整和集成的数据，才能进行深层次的分析，寻求有益的信息，而这些正是数据仓库所能提供的。数据仓库不仅是集成数据的一种方式，数据仓库的联机分析功能 OLAP 还为数据挖掘提供了一个极佳的操作平台。如果数据仓库与数据挖掘能够实现有效的协同，将给数据挖掘带来各种便利和功能。数据挖掘也为数据仓库提供更好的决策支持，促进了数据仓库技术的发展。

4. 数据挖掘技术的应用过程

数据挖掘过程一般需要经历确定挖掘对象、准备数据、建立模型、数据挖掘、结果分析与知识应用这样几个阶段。

（1）确定挖掘对象

数据挖掘的第一步是要定义清晰的挖掘对象，认清数据挖掘的目标。数据挖掘的最后

结果往往是不可预测的，但探索的问题应是有预见性的、有目标的。为了数据挖掘而挖掘数据带有盲目性，往往是不会成功的。在定义挖掘对象时，需要确定这样的问题：从何处入手？需要挖掘什么数据？要用多少数据？数据挖掘要进行到什么程度？尽管在数据挖掘中常常事先不能确定最后挖掘的结果到底是什么。

有时还要用户提供一些先验性知识，如概念树等。这些先验性知识可能是用户业务领域知识或以前数据挖掘所获得的初步成果。这就意味着数据挖掘是一个过程，在挖掘过程中可能提出新的问题，可能尝试用其他方法来检验数据，在数据的子集上进行同样的研究。有时业务对象是一些已经理解的数据，但在某些情况下还需要对这些数据进行挖掘。此时，不是通过数据挖掘发现新的有价值的信息，而是通过数据挖掘验证假设的正确性，或者是通过同样方式的数据挖掘查看模式是否发生变化。如果在经常性的同样的数据挖掘中的一次挖掘没有出现以前同样的结果，这意味着模式已经发生了变化，可能需要进行更深层次的挖掘。例如，将数据挖掘应用于客户关系管理（CRM）中，就需要对客户关系管理的商业主题进行仔细的定义。每个 CRM 应用都有一个或多个商业目标，要为每个目标建立恰当的模型。比如，“提高客户对企业促销的响应率”和“提高每个客户的响应价值”这两个目标是不同的，并且在定义问题的同时，也生成了评价 CRM 应用结果的标准和方法，即确定了数据挖掘的评价指标。

（2）准备数据

在确定数据挖掘的业务对象后，需要搜索所有与业务对象有关的内部和外部数据，从中选出适合于数据挖掘应用的数据。对数据的选择必须在建立数据挖掘模型之前完成。选择数据后，还需要对数据进行预处理，并对数据进行清洗，解决数据中的缺值、冗余、数据值的不一致性、数据定义的不一致性以及过时数据等问题。在数据挖掘时，有时还需要对数据分组，以提高数据挖掘的效率，降低模型的复杂度。

（3）建立模型

将数据转换成一个分析模型，这个分析模型是针对挖掘算法建立的。建立一个真正适合挖掘算法的分析模型，是数据挖掘的关键。

（4）数据挖掘

对所得到的经过转化的数据进行挖掘，除了完善与选择合适的算法需要人工干预外，数据挖掘工作都由数据挖掘工具自动完成。

（5）结果分析

当数据挖掘出现结果后，要对挖掘结果进行解释和评估。具体的解释和评估方法一般应根据针对数据挖掘结果所制定的决策成败而定，但是管理决策分析人员在使用数据挖掘结果之前，又希望能够对挖掘的结果进行评估，以保证数据挖掘结果在实际应用中的成功率。因此，在对数据挖掘结果进行评价时，可以考虑这样几个方面的问题：①建立模型，相同的数据集在模型上进行操作所获得的结果要优于用不同数据集在模型上的操作结果；②模型的某些结果可能比其他预测结果更加准确；③由于模型是以样板数据为基础建立的，因此实际结果往往会比建模时的结果差。另外，利用可视化技术可将数据挖掘结果表现得更清楚，更有利于对数据挖掘结果的分析。

（6）知识应用

数据挖掘的结果需要经过业务决策人员的认可，才能实际利用。要将通过数据挖掘得

出的预测模式和各个领域的专家知识结合在一起，构成一个可供不同类型的人使用的应用程序。也只有通过对挖掘知识的应用，才能对数据挖掘的成果做出正确的评价。但是，在应用数据挖掘的成果时，决策人员关心的是数据挖掘的最终结果与用其他候选结果在实际应用中的差距。以往在进行较复杂的数据分析时，专家们限于时间因素，不得不对参加运算的变量和数量加以限制，但是那些被丢弃而没有参加运算的变量有可能包含着另一些不为人知的有用信息。现在，高性能的数据挖掘工具让用户能对数据库进行通盘的深度遍历，并且把任何可能参选的变量都考虑进去，再也不需要选择变量的子集来进行运算了。广度上，允许有更多的行存在。更大的样本使得产生错误和变化的概率降低，这样用户就能更加精确地推导出一些虽小但颇为重要的结论。

10.5 其他新型的数据库系统

10.5.1 多媒体数据库系统

近年来，随着多媒体数据的引入，对数据管理方法又开始酝酿新的变革。传统的数据库所处理的是字符数值类数据，应用现有的数据库理论和方法可把现实世界抽象为数据模型，即实现数据的规范化，并可完成对数据的管理。当图形、图像、声音及视频等多媒体信息引入计算机之后，可表达的信息范围大大扩展。由于多媒体数据不规则，有丰富的信息含量、复杂的结构及关系，并有巨大的数据量，所以传统的数据库技术如数据类型、数据模型、操作语言、存储结构、存取路径、检索机制以及网络和数据传递等都不能适应复杂应用对象的应用需求，这就需要一种新的管理系统对多媒体数据进行有效的组织、管理和存取。这种把多媒体技术和数据库相结合的数据库系统称为多媒体数据库系统。多媒体数据库系统是除了文本和其他离散数据外，还能够存储、操作和检索音频和视频等连续数据的数据库系统。可以实现以下功能：多媒体数据库对象的定义；多媒体数据存取；多媒体数据库进行控制；多媒体数据组织、存储和管理；多媒体数据库建立和维护；多媒体数据库在网络上的通信功能。可以说，多媒体数据库是数据库技术与多媒体技术相结合的产物。与传统的数据库相比，多媒体数据库具有以下特点。

1. 存储和处理的信息量大

常规数据的数据量较小，而多媒体数据的数据量巨大，两者之间的差别可大到几千、几万甚至几十万倍。多媒体功能要求对分布在不同辅助存储媒体上的海量信息进行数据库的管理。如果在分布式应用中，数据需要通过网络进行存取，庞大的数据量对数据库技术是一种严峻的考验。

2. 数据长度不确定

常规数据在组织存储时一般长度可确定，可用定长记录来存储，因而可以构造成一张张的二维表，每张表即对应一个关系，每一行表示一个元组，每一列表示一个属性，每个数据都是不可再分的原子数据，存取方便，结构简单。而多媒体数据的数据量大小不定，长度难以预测，导致数据不能用定长来存储，数据存储过程比较麻烦。

3. 要求实现同步性、实时性

连续数据流对多媒体数据库提出了实时性处理的要求。多媒体数据，无论是声音媒体还是视频媒体，都要求连续传送或输出，否则将导致严重失真，影响效果。这就要求多媒体数据库系统在技术上实现同步性、实时性。

4. 数据定义及操作不同

传统的关系数据库处理的是规范关系，每个元组由定长的属性值组成，而每个属性值又是不可再分的原子数据。因而，对这些规范关系可方便地定义并施行各种标准操作，从而可为用户提供简明的数据视图以及使用简单方便而功能强大的 SQL 语言。而多媒体数据不规则，有丰富的信息含量、复杂的结构及关系，所以无论是描述语言、数据操作语言或存储结构、存取路径等都和传统的关系数据库有所不同。

5. 数据复杂性增大

由于不同媒体之间的特性差异很大，因此增加了数据库在管理和处理这些媒体数据时的复杂性。通常的多媒体数据库中，音频、视频、图像都存在，为了管理方便，应尽可能把上述三种媒体存放在不同的表中，然后通过外键进行连接。

10.5.2 空间数据库系统

1. 空间数据

所谓空间数据是指与空间位置和空间关系相联系的数据。每个空间对象都具有空间坐标，即空间对象隐含了空间分布特征。这意味着在空间数据组织方面，要考虑它的空间分布特征。空间数据量比一般的通用数据库要大得多，通常称为海量数据。正因为空间数据量大，所以需要在二维空间上划分块或者图幅，在垂直方向上划分层来进行组织。空间数据通常具有非结构化的特征。在一般的关系数据库管理系统中，数据记录一般是结构化的，即它满足关系数据模型的第一范式要求，每一条记录都是定长的，数据项表达的只能是原子数据，不允许嵌套记录。而空间数据则不能满足这种结构化要求。若用一条记录表达一个空间对象，它的数据项可能是变长的，它不满足关系数据模型的范式要求，这也是空间图形数据难以直接采用一般的关系数据管理系统的主要原因。

2. 空间数据库系统

空间数据库是关于某一区域内一定地理要素特征的数据集合，是地理信息统中空间数据的存储场所，在整个地理信息系统中占有极其重要的地位，是地理信息系统发挥功能和作用的关键。空间数据库以地理空间数据存储和操作为对象，把被管理的数据从一维推向了二维、三维甚至更高维。由于传统数据库系统的数据模型主要针对简单对象，因而无法有效地支持以图形、影像等复杂对象为主体的工程应用。空间数据库系统的形成和发展，是数据存储和管理的需要，也是传统数据库系统在空间的扩展。从概念上讲，空间数据库系统是用于采集、存储、管理、检索、分析和表达空间数据和属性数据的计算机系统，能够对海量数据

进行存储和管理。空间数据库是随着地理信息系统的开发和应用发展起来的数据库新技术。空间数据库系统应用紧密结合，大多数以地理信息系统的基础和核心的形式出现。空间数据库系统必须具备对地理对象进行模拟推理的功能。一方面可将空间数据库技术视为传统数据库技术的扩充。空间数据库系统的扩充部分包括空间数据模型、空间关系、空间查询语言、空间索引、空间查询处理与优化、空间数据库实现技术、并行分布式空间数据库系统和空间数据仓库。另一方面，空间数据库突破了传统数据库理论，其实质性发展必然导致理论上的创新。

10.5.3　模糊数据库系统

1965 年 Zadeh 提出模糊集合理论，这对几乎所有的数学分支都产生了重要影响，应用遍及各个领域。在数据库理论与技术中使用模糊理论，开始于 1979 年 Buckles 等人建立的模糊数据库理论，而国际上对模糊数据库的研究主要是从 20 世纪 80 年代开始的，旨在克服传统数据库难以表达和处理模糊信息的特点，进而扩展数据库的功能，开拓更新更广的领域。

传统数据库仅允许对精确数据进行存储和处理，而客观世界中有许多事物是不精确的。模糊数据库的研究和实践就是为了解决模糊数据的表达和处理问题，使得数据库描述的模型能更自然、更贴切地反映客观世界。从概念上讲，模糊数据库是指能够处理模糊数据的数据库，它是研究如何在数据中表示、存储和管理模糊数据的一门学科，也是数据库领域的一个重要分支。与传统数据库不同，模糊数据库中数据本身是模糊的，包括模糊数据变量和模糊语言变量。数据间的联系以及依赖关系也是模糊的，数据完整性、一致性等约束条件以及数据定义、数据操纵和数据查询等都具有模糊性。实现模糊数据库，要研究模糊算法、模糊逻辑和模糊推理等方面的理论、方法和技术问题具有一定的困难。由于关系数据库是数据库逻辑模型中最简单的，因此最早研究出来的都是一些模糊关系型数据库的数据模型。此后大量的工作集中在模糊关系数据库方面，即对关系数据库进行模糊扩展，提出了各种模糊关系数据模型、模糊关系数据库语言模型和模糊关系数据库的物理组织形式等，并对传统数据库中的相关概念进行了新的定义，如模糊关系、模糊数据字典、模糊运算符和模糊操作符等。

近年来，模糊数据库的重心从模糊性扩展到关系数据库模型的理论框架研究，如模糊网络数据模型、模糊层次数据模型、模糊关系数据模型、面向对象的模糊数据模型、模糊逻辑数据模型、模糊演绎数据模型、模糊知识库模型等。模糊数据库与专家系统、知识库等技术的结合也是模糊数据库技术的研究方向。

10.5.4　智能数据库系统

智能数据库是数据库技术和人工智能技术相结合的产物。从人工智能领域寻求和吸收新的思想、理论和方法，将其与传统数据库理论与技术相结合，推出新的数据模型，建立和实现新一代的数据库系统——智能数据库。智能数据库研究利用人的推理、想象、记忆原理，实现对数据库的存储、搜索和修改。智能数据库通过有效的组织，能够满足人们快速检索和修改数据库的要求。智能数据库思想的提出，预示着人类的信息处理即将步入一个崭新的时代。

1. 人工智能

人工智能(Artificial Intelligence,AI)是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。人工智能是计算机科学的一个分支,也被认为是21世纪世界三大尖端技术之一。

"人工智能"一词最初是在1956年Dartmouth学会上提出的。从那以后,研究者们发展了众多理论和原理,人工智能的概念也随之扩展。美国斯坦福大学人工智能研究中心的尼尔逊教授对人工智能给出了如下定义:人工智能是关于知识的学科——怎样表示知识以及怎样获得知识并使用知识的科学。美国麻省理工学院的温斯顿教授认为:"人工智能就是研究如何使计算机去做过去只有人才能做的智能工作。"

这些说法反映了人工智能学科的基本思想和基本内容,即人工智能是研究人类智能活动的规律,构造具有一定智能的人工系统,研究如何让计算机去完成以往需要人的智力才能胜任的工作,也就是研究如何应用计算机的软硬件来模拟人类某些智能行为的基本理论、方法和技术。

可以说,人工智能是一门极富挑战性的边缘学科,除了计算机科学以外,人工智能还涉及信息论、控制论、自动化、仿生学、生物学、心理学、数理逻辑、语言学、医学和哲学等多门学科。总的说来,人工智能研究的一个主要目标是使机器能够胜任一些通常需要人类智能才能完成的复杂工作。

2. 智能数据库

智能数据库是数据库技术和人工智能技术相结合的产物。智能化的数据库是将人工智能技术与传统数据库理论与技术相结合,推出新的数据模型,使其不但具有传统数据库的现有功能,还扩展了如下一些AI能力,以提高数据库的演绎、推理功能和智能化的程度。

(1) 数据库的演绎功能

利用AI技术,能从现有数据库的数据中演绎和推导出一些新数据。

(2) 数据库的搜索功能

将数据库中的操作视为AI中的问题求解——搜索,利用人工智能算法,实现数据库的智能检索和查询优化。

(3) 数据库的问题求解能力

将知识表示和知识处理为主的专家系统技术与数据管理为主的数据库技术相结合,开发出能共享信息的面向知识处理的问题求解系统。

(4) 数据库的归纳功能

利用AI技术,将数据库中的数据归纳成规则,存入知识库,进一步发展即可使数据库具有学习能力。

(5) 数据库的知识管理能力

在数据库只能管理事实数据的基础上,再加以扩充,使其具有管理事实与规则的功能,即管理知识的能力。

智能数据库将计算机科学中近年来日趋发展成熟的几种主要技术,面向对象技术、数据库技术、人工智能集成为一体。智能数据库系统是一个对象数据库管理系统。体系结构的

选择对它的性能和功能有非常重要的影响。在体系结构的选择过程中我们始终遵循这样的一条准则：性能更为重要。因为功能可以在ODBMS上层的应用程序中增加，而性能上的缺陷是不可能在应用程序层次上得以弥补的。

10.6　本章小结

本章介绍数据库技术的新发展，其中，面向对象数据库是数据库技术与面向对象程序设计方法的结合；分布式数据库是数据库技术与网络通信技术的结合；Web数据库是数据库技术与Web技术的结合；数据仓库是数据库技术与人工智能技术的结合。另外，还介绍了面向特定应用领域的一些新型数据库系统。

10.7　习题

10.7.1　名词解释

面向对象数据模型、分布式数据库系统、并行数据库系统、数据仓库、数据挖掘

10.7.2　简答题

1. 简述面向对象数据库系统的基本特征。
2. 简述分布式数据库的特点。
3. 简述数据仓库的基本特性。
4. 简述数据挖掘技术的应用过程。

参 考 文 献

1 王珊,萨师煊.数据库系统概论.4版.北京:高等教育出版社,2006.

2 Peter Rob,Carlos Coronel 著.数据库系统设计实现与管理.张瑜,等译.6版.北京:清华大学出版社,2005.

3 Hoffer J A,Prescott M B,Mcfadden F R.现代数据库管理.8版.北京:清华大学出版社,2008.

4 王亚平主编.数据库系统工程教程.北京:清华大学出版社,2004.

5 [美]沃尔曼等著.数据库系统基础教程.岳丽华,龚育昌,等译.北京:机械工业出版社,2003.

6 西尔伯沙茨等著.数据库系统概念.杨冬青,等译.5版.北京:机械工业出版社,2006.

7 闪四清.SQL Server 实用简明教程.北京:清华大学出版社,2006.

8 Jeffrey A Hoffer,Mary B Prescott,Fred R McFadden 著.现代数据库管理.袁方,等译.7版.北京:电子工业出版社,2006.

9 PaulrajPonniah 著.数据库设计与开发教程.韩宏志译.北京:清华大学出版社,2005.

10 Abraham Silberschatz, Henry F Korth, S Sudarsham. Database System Concepts, Fifth Edition. Published by McGraw-H ill,2005.

11 Gerald V Post. Database Management Systems. Designing & Building Business Applications.影印版.北京:机械工业出版社,2006.

12 Philip M Lewis,Arthur Berntein,Michael Kifer 著.数据库与事务管理.施伯乐,方锦成,等译.北京:机械工业出版社,2005.

13 Gerald V Post 著.数据库管理系统.冯建华,刘旭辉,周维续,等译.北京:机械工业出版社,2006.

14 王亚平.数据库系统原理辅导.西安:西安电子科技大学出版社,2003.

15 钱雪忠,黄学光,刘肃平.数据库原理及应用.北京:北京邮电大学出版社,2005.

16 苏俊.数据库基础教程.北京:中国人民大学出版社,2002.

17 冯建华.数据库系统设计与原理.北京:清华大学出版社,2005.

18 陈雁.数据库系统原理与设计.北京:中国电力出版社,2003.